

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-112901

(43)Date of publication of application : 21.04.2000

(51)Int.Cl.

G06F 15/16
G06F 9/45

(21)Application number : 10-281692

(71)Applicant : HITACHI LTD
REAL WORLD COMPUTING
PARTNERSHIP

(22)Date of filing : 02.10.1998

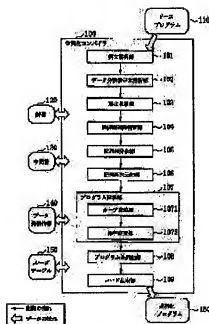
(72)Inventor : SATO MAKOTO
HIROOKA TAKASHI

(54) DATA REARRANGING METHOD

(57)Abstract:

PROBLEM TO BE SOLVED: To eliminate the need of a copy in data re-distribution while a program is executed and to speed up the execution of the program in a distributed storage-type or distributed common storage-type parallel machine.

SOLUTION: A data distribution form showing the allocation method of respective array elements to a distributed memory is collected (form collection part 103). Respective arrays are divided into sets formed of partial arrays belonging to a unique processor in any data distribution form (array thinning part 105). The number of the dimensions of the original array is increased so that the array elements in the respective partial arrays become continuous addresses and the order of the array elements is rearranged (array high-dimension forming part 106). It is judged that the number of the types of the data distribution forms in the respective dimensions of array is not more than two (form type number judgement 104). It is detected that a loop control variable is contained in the added character of the dimension which is made to be higher in array and a loop having the loop control variable is converted into plural loops nested in accordance with the forming of the array dimension to be high (loop conversion part 1071). The array added character in an array reference point in a program is changed in accordance with the forming of the array dimension of array added character at an array reference point in a program to be high (added character change part 1072).



(51)IntCl. ⁷	識別記号	F I	マークシート(参考)
G 0 6 F 15/16 9/45	6 2 0	G 0 6 F 15/16 9/44	6 2 0 H 5 B 0 4 5 3 2 2 F 5 B 0 8 1

審査請求 未請求 請求項の数 8 O L (全 34 頁)

(21) 出願番号 特願平10-281692

(22) 出願日 平成10年10月2日(1998.10.2)

(71) 出願人 000005108

株式会社日立製作所
東京都千代田区神田駿河台四丁目6番地

(71) 出願人 583162453

技術研究組合新情報処理開発機構
東京都千代田区東神田2-5-12 龍角散ビル8階

(72) 発明者 佐藤 真平

神奈川県川崎市麻生区王禅寺1099番地 株式会社日立製作所システム開発研究所内

(74) 復代理人 100102587

弁理士 渡邊 昌幸 (外1名)

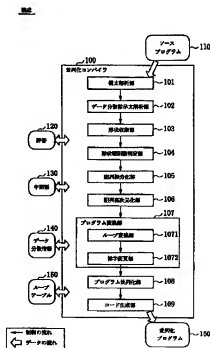
最終頁に続く

(54) 【発明の名称】 データ再配置方法

(57) 【要約】

【課題】 分散記憶型または分散共有記憶型並列機で、プログラム実行中のデータ再分散におけるコピーを不要にしプログラムの実行を高速化すること。

【解決手段】 各配列要素の分散メモリへの割付け方法を表わすデータ分散形状を収集し(形状収集部103)、各配列をどのデータ分散形状においても各部分配列が唯一のプロセッサに属するような部分配列からなる集合に分割し(配列細分化部105)、各部分配列内の配列要素が連続アドレスになるように元の配列の次元数を増加させて配列要素の順序を並び替え(配列高次元化部106)、配列の各次元のデータ分散形状の種類が2種類以下であることを判定し(形状種別数判定104)、配列の高次元化される次元の添字にループ制御変数が含まれることを検出し、そのループ制御変数を持つループを配列高次元化に対応してネストする複数のループに変換し(ループ変換部1071)、プログラム中の配列参照点における配列添字を配列高次元化に対応して変更する(添字変更部1072)。



【特許請求の範囲】

【請求項1】 逐次計算機向けのソースプログラムまたはデータ分散指示文あるいはデータ再分散指示文を含む並列計算機向けのソースプログラムを入力し、並列計算機の各分散メモリにデータを分散し、データ再分散指示文またはデータ再分散ルーチンを含む並列計算機向けソースプログラムまたはオブジェクトプログラムを出力するコンパイラにおけるデータ再配置方法であって、プログラム中の各点における、各配列要素の分散メモリへの割付け方法を表わすデータ分散形状を収集する形状

収集ステップと、各配列を、上記形状収集ステップで得られたどのデータ分散形状においても、各部分配列が唯一のプロセッサに属するような部分配列からなる集合に分割する配列細分

化ステップと、上記各部分配列内の配列要素が連続アドレスになるように元の配列の次元数を増加させて配列要素の順序を並び替える配列高次元化ステップを有することを特徴とするデータ再配置方法。

【請求項2】 逐次計算機向けまたは並列指示文を含む並列計算機向けのソースプログラムを入力し、共有メモリ型並列計算機向けの並列化指示文を含むソースプログラムまたはオブジェクトプログラムを出力するコンパイラにおけるデータ再配置方法であって、

複数のプロセッサがその部分配列を分担して計算すべき配列に対して、プログラム中の各点における、各配列要素のプロセッサへの割付け方法を表わすデータ分担形状を収集する形状収集ステップと、

各配列を、上記形状収集ステップで得られたどのデータ分担形状においても各部分配列が唯一のプロセッサの分担になるような部分配列からなる集合に分割する配列細分

化ステップと、上記各部分配列内の配列要素が連続アドレスになるように元の配列の次元数を増加させて配列要素の順序を並び替える配列高次元化ステップを有することを特徴とするデータ再配置方法。

【請求項3】 請求項1または2記載のデータ再配置方法において、上記形状収集ステップによって得られた配列の各次元のデータ分散形状またはデータ分担形状の種類が2種類以下であることを判定する形状種別数判定ステップをさらに有することを特徴とするデータ再配置方法。

【請求項4】 請求項1または2記載のデータ再配置方法において、上記配列高次元化ステップは、元の配列の各要素を、それぞれの請求項における配列細分化ステップによって得られた部分配列中の要素と各部分配列の全体配列中での位置を示す位置配列の要素との組で表現可能なように、元の配列から、上記部分配列の次元数と上記位置配列の次元数の和の次元数を持つ高次元化配列を作成し、上記高次元化配列の連続アドレスを与える次元

から上記部分配列の次元数分の次元までの各寸法は、上記部分配列の各次元の寸法とし、残りの次元の寸法は、上記位置配列の各次元の寸法とすることを特徴とするデータ再配置方法。

【請求項5】 請求項1または2記載のデータ再配置方法において、上記配列高次元化ステップに伴い、配列の高次元化された次元の添字にループ制御変数が含まれることを検出し、そのループ制御変数を持つループを、配列高次元化ステップに対応してネストする複数のループに変換するループ変換ステップと、プログラム中の配列参照点における配列添字を、配列高次元化ステップに対応して変更する添字変更ステップとをさらに含むことを特徴とするデータ再配置方法。

【請求項6】 請求項5記載のデータ再配置方法において、上記ループ変換ステップは、配列要素の参照が部分配列間の境界をまたがるか否かを判定し、境界をまたがる場合とまたがらない場合とで別ループになるようにループの繰返し範囲を分割するための解析を行なうループピーリング解析ステップと、上記解析の結果を用いてループの繰返し範囲を分割するループピーリングステップとを含むことを特徴とするデータ再配置方法。

【請求項7】 請求項1記載のデータ再配置方法において、通信データのバッファリングを行わない再分散通信を生成する非バッファリング再分散通信生成ステップをさらに含むことを特徴とするデータ再配置方法。

【請求項8】 請求項7記載のデータ再配置方法において、分散メモリ型並列計算機の各プロセッサで、データ分散前の元の配列全体を宣言するグローバルデータ宣言作成ステップと、各配列参照点における配列添字の値を、データ分散前の元の値を用いるグローバル添字生成ステップをさらに含むことを特徴とするデータ再配置方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、逐次計算機向けのプログラムまたは並列計算機向けのデータ再分散指示を含むプログラムを入力してデータ再配置を行ったり、あるいは、データ再分散指示を挿入した並列計算機向けのプログラムまたはオブジェクトプログラムを生成するコンパイラに係り、特に、バッファへのコピー時間を不要とし、プログラムの実行時間を短縮することが可能なデータ再配置方法に関する。

【0002】

【従来技術】従来、分散メモリ型並列計算機におけるデータ再分散を高速化する技術として、ダニエル・ジェイ・パレルモ、オイグネ・ヴェー・ホッジ・フィリス、ブリス・ヴィラージ・バナジー著「ダイナミック データパーティショニング フォー ディストリビューテッドメモリ マルチコンピュータズ」ジャーナル オブ パラレル アンド ディストリビューテッド コンピューティ

ング 38、第158頁から第175頁(1988)(Daniel J. Palermo, Eugene W. Hodges IV, and Prithviraj Banerjee, "Dynamic Data Partitioning for Distributed-Memory Multicomputers", Journal of Parallel and Distributed Computing 38, pp. 158-175 (1998))中のpp.170-171で言及されているように、データ再分散オペレーションのアップレーション、即ち、データ再分散すべき複数の配列、または、データ再分散すべき一つの配列の中の非連続な部分を、一つのバッファの連続アドレス領域へプログラム実行時にコピーし、そのバッファに対して再分散を唯1回、行なうことで、再分散の起動オーバーヘッドを削減するものがあった。

【0003】また、従来、共有メモリ型並列計算機におけるFalse Sharingと呼ばれるキャッシュ間コンフリクトを削減する技術として、ジェニファー・エム・アンダーソン、サマン・ビー・アマラシンハ、モニカ・エス・ラム著「データ アンド コミュニケーション トランスフォーメーションズ フォー マルチプロセッサズ」ピーイーオービー95、第166頁から第178頁、1995(Jennifer M. Anderson, Saman P. Amarasinghe, and Monica S. Lam, "Data and Communication Transformations for Multiprocessors", PPOPP'95, pp. 166-178 (1995))で述べられているように、配列の次元の内、アドレスが連続する配列要素を特定するのに使われる次元を、プロセッサ個数に分割し、その次元を、一つのプロセッサがアクセスする添字だけとなる次元と、プロセッサ番号を指定する次元の2つの次元に分割し、後者を、アドレスの増加が最も遅い次元になるように配列を高次元化することで、キャッシュ内のデータが2つのプロセッサにまたがる機会を減少させるものがあった。

【0004】

【発明が解決しようとする課題】上記第1の従来技術は、プログラム実行時にバッファへコピーするため、コピーに余分な時間がかかるという問題点がある。また、上記第1の従来技術は、再分散前後で配列の大きさが変化する可能性があるため、再分散のたびに配列領域の確保・開放が必要で、それらに余分な時間がかかるという問題点がある。さらに、上記第1の従来技術は、上記配列領域の確保・開放、および、再分散前後で配列の形状が変化するにより、本来は不要な、再分散前後で同一のプロセッサに割り当てられる部分配列のコピーも必要で、それらに余分な時間がかかるという問題点がある。

【0005】上記第2の従来技術は、ある特定の次元の配列要素を複数プロセッサが分担して計算する時には適用可能だが、配列が3次元以上で、プログラム中に2つの異なるループが存在し、複数プロセッサが分担して計算する配列の次元がそれらのループに対して、各々、アドレスが連続になる次元から1次元目と2次元目である時、前者のループに対してはキャッシュ間コンフリクト

の一種であるフォールスシェアリングがかなり削減可能である。しかし、後者のループに対しては配列の2次元目の、プロセッサ間の割り当て境界で、3次元目の各添字に対してプロセッサ総数だけ両プロセッサがキャッシュ共有する可能性があるため、フォールスシェアリングはあまり削減できないという問題点がある。

【0006】本発明の目的は、プログラム実行時にバッファへのコピー処理をなくし、プログラムの実行時間を短縮することである。また、本発明の別の目的は、再分散時の、配列領域の確保・開放処理をなくし、プログラムの実行時間を短縮することである。また、本発明の別の目的は、再分散前後で同一のプロセッサに割り当てられる部分配列のコピーをなくすことにより、プログラムの実行時間を短縮することである。また、本発明の別の目的は、プログラム中に複数の異なるループが存在し、各々のループで、複数プロセッサが分担して計算する配列の次元が異なる時に、それら全てのループに対してキャッシュ間コンフリクトを削減することにより、プログラムの実行時間を短縮することである。

【0007】

【課題を解決するための手段】本発明は、上記目的を達成するために、プログラム中の各点における、各配列要素の分散メモリへの割付け方法を表すデータ分散形状を収集する形状収集ステップ(図1の形状収集部103)と、各配列を、上記形状収集で得られたデータ分散形状においても、各部分配列が唯一のプロセッサに属するような部分配列からなる集合に分割する配列細分化ステップ(同、配列細分化部105)と、上記各部分配列内の配列要素が連続アドレスになるように元の配列の次元数を増加させて配列要素の順序を並び替える配列高次元化ステップ(同、配列高次元化部106)と、配列の各次元のデータ分散形状の種類が2種類以下であることを判定する形状種別数判定ステップ(同、形状種別数判定104)と、配列の高次元化される次元の添字にループ制御変数が含まれることを検出し、そのループ制御変数を持つループを、配列高次元化に対応して、ネストする複数のループに変換するループ変換ステップ(同、ループ変換部1071)と、プログラム中の配列参照点における配列添字を、配列高次元化に対応して変更する添字変更ステップ(同、添字変更部1072)と、分散メモリ型並列計算機の各プロセッサで、データ分散前の元の配列全体を宣言するグローバルデータ宣言作成ステップ(図7のステップ706)と、各配列参照点における配列添字の値を、データ分散前の元の値を用いるグローバル添字生成ステップ(図7のステップ7052)と、通信データのバッファリング再分散通信生成ステップ(図7のステップ7041)を含むものである。【0008】また、本発明は、上記別の目的を達成するために、複数プロセッサがその部分配列を分担して計算

すべき配列に対して、プログラム中の各点における、各配列要素のプロセッサへの割付け方法を表すデータ分担形状を収集する形状収集ステップ(図27の形状収集部103)と、各配列を、上記形状収集で得られたどのデータ分担形状においても、各部分配列が唯一のプロセッサの分担になるような部分配列からなる集合に分割する配列細分化ステップ(同、配列細分化部105)と、上記各部分配列内の配列要素がアドレスになるように元の配列の次元数を増加させて配列要素の順序を並び替える配列高次元化ステップ(同、配列高次元化部106)と、配列の各次元のデータ分担形状の種類が2種類以下であることを判定する形状種別数判定ステップ(同、形状種別数判定部104)と、配列の高次元化される次元の添字にループ制御変数が含まれることを検出し、そのループ制御変数を持つループを、配列高次元化に対応して、ネストする複数のループに変換するループ変換ステップ(同、ループ変換部1071)と、プログラム中の配列参照点における配列添字を、配列高次元化に対応して変更する添字変更ステップ(同、添字変更部1072)と、分散メモリ型並列計算機の各プロセッサで、データ分散前の元の配列全体を宣言するグローバルデータ宣言作成ステップ(図7のステップ706)と、各配列参照点における配列添字の値を、データ分散前の元の値を用いるグローバル添字生成ステップ(図7のステップ7052)と、通信データのバッファリングを行なわない再分散通信を生成する非バッファリング再分散通信生成ステップ(図7のステップ7041)を含むものである。

【0009】

【発明の実施の形態】(第1の実施例)以下、本発明の第1の実施例を図1から図26を用いて説明する。第1の実施例では分散メモリを持つ並列計算機を対象としている。図1は、本発明による並列化コンパイラの構成を示したものである。100は並列化コンパイラである。並列化コンパイラ100は、同図に示すように、ソースプログラム110を入力して辞書120及び中間語130を出力する構文解析部101、辞書120及び中間語130を入力してデータ分散指示文を解析し、データ分散情報140を出力するデータ分散指示文解析部102、データ分散情報140を入力してプログラム内に現れるデータ分散形状情報を収集し、その結果を辞書120に反映させる形状収集部103、辞書120及びデータ分散情報140を入力して配列のデータ分散の種別数を判定し、配列を細分化するべきか否かを辞書120に反映させる形状種別数判定部104、辞書120及びデータ分散情報140を入力して配列を細分化し、その結果を辞書120に反映させる配列細分化部105、辞書120及びデータ分散情報140を入力して、データ分散形状が変化する次元の配列添字範囲を分割して、その次元を2つの次元に分けることによって配列を高次元化

し、その結果を辞書120に反映させる配列高次元化部106、辞書120、データ分散情報140及び中間語130を入力してプログラムを変換し、その結果を中間語130に反映し、またループテーブル150を出力するプログラム変換部107、辞書120、データ分散情報140、中間語130及びループテーブル150を入力してプログラムを並列化し、その結果を辞書120、中間語130、ループテーブル150に反映するプログラム並列化部108、辞書120、中間語130、ループテーブル150を入力して並列化プログラム160を出力するコード生成部109よりなる。

【0010】ここで、プログラム変換部107は、辞書120、データ分散情報140及び中間語130を入力して、配列を高次元化する時に添字範囲を分割することによって生じる、分割境界をまたいだ配列要素の参照とそうでない配列要素の参照で添字が異なることに対処するために、分割境界をまたいだ配列要素を参照する文を含むループ繰り返しと、そのような文を含まないループ繰り返しを別のループにするループピーリングと、配列高次元化に伴い、増加した添字に対するループを作成するループ多重化とを行い、その結果をループテーブル150を作成してそこに反映し、また中間語130に反映するループ変換部1071、配列の高次元化とループピーリングに伴って、配列の添字を変更する添字変更部1072、よりなる。

【0011】本実施例におけるソースプログラム110は、H P F (High Performance Fortran) のようなデータ分散指示文の入ったプログラムであり、並列化プログラム160は、分散メモリマシン向けの M P I (Message Passing Interface) のようなメッセージ通信入りの並列化プログラムである。ここで、データ分散指示文は、データを分割して分割された個々のデータを複数の論理プロセッサの内の一つに割り付けるように、コンパイラに指示する文である。

【0012】H P F の言語仕様の説明は、ハイ パフォーマンス フォートラン フォラム編「ハイ パフォーマンス フォートラン ランゲージ スペシフィケーション バージョン2.0.a.2」センター フォー リサーチ オン パラレル コンピューテーション、ライズ ユニバーシティ、ヒューストン、テキサス、1996 (High Performance Fortran Forum, "High Performance Fortran Language Specification Ver. 2.0.a.2", Center for Research on Parallel Computation, Rice Univ., Houston, Tx, 1996.) に詳しい。

【0013】M P I の仕様の説明は、メッセージ パッシング インターフェース フォーラム編「エム ピーアイ: ア メッセージ-パッシング インターフェース スタンダード」ユニバーシティ オブ テネシー、タックスビル、テネシー、1994 (Message Passing Interface Forum, "MPI: A Message-Passing Interface Standar

d", University of Tennessee, Knoxville, Tennessee e, 1994.) に詳しい。

【0014】次に、本発明の並列化コンパイラの動作を、図1を元に、図2から図26まで、具体例を用いて説明する。図8はソースプログラム110の具体例である。文800は、Fortranの文法による配列a及びスカラー変数b、cの宣言である。文803と文817で囲まれた部分は、kをループ制御変数とするループ本体である。文805と文809で囲まれた部分は、jをループ制御変数とするループ本体であり、文806と文808で囲まれた部分は、iをループ制御変数とするループ本体である。文811と文815で囲まれた部分も同様に、jをループ制御変数とするループ本体であり、文812と文814で囲まれた部分も同様に、iをループ制御変数とするループ本体である。

【0015】文816は、スカラー変数b、cの値を変更する手続きを呼出し文である。文801、文802、文804、及び文810はHPFの指示文である。“!HPFS”は、それで始まる文がHPFの指示文であることを表すキーワードである。文801は論理プロセッサの配置形状を指定する指示文である。“PROCESSORS”はキーワードであり、それに続く“proc (4)”により、論理プロセッサ4台が直線状に配置されていることを宣言している。文802は、配列aの初期分割を指示する文である。“DISTRIBUTE”はキーワードであり、それに続く“a (block, *) ONTO proc”により、配列aの1次元目をblockという方法で分割し、分割された各々の部分配列を論理プロセッサproc (1)、proc (2)、proc (3)、proc (4)に、それぞれ割り付けることを表す。

【0016】ここで、上記の論理プロセッサ宣言において、配列aの1次元目をblock分割するとは、配列aの1次元目の下限値を1、上限値をNとすると、配列aの1次元目の添字を、1から[(N-1)/4]+1個ずつ区切って、proc (1)から順番に割り付けることである。今の場合、a (1:3, 1:12), a (4:6, 1:12), a (7:9, 1:12), a (10:12, 1:12)がそれぞれ、proc (1)、proc (2)、proc (3)、proc (4)に割り付けられる。

【0017】文804は配列aをプログラム実行中に再分割することを指示する文である。“REDISTRIBUTE”はキーワードであり、それに続く“a (block, *) ONTO proc”により、配列aの1次元目をblockという方法で分割し、分割された各々の部分配列を論理プロセッサproc (1)、proc (2)、proc (3)、proc (4)に、それぞれ割り付けることを表す。この指示文により、それまで他の方法で論理プロセッサに割り付けられていた配列

要素は、配列aの1次元目をblock分割するような割り付けられ方に変更される。これにより、一般に論理プロセッサ間のデータ移動が発生する。

【0018】文810も同様に配列aをプログラム実行中に再分割することを指示する文である。この文は、文804とは異なり、配列aの2次元目をblock分割するような割り付け方を指示する。今の場合、a (1:12, 1:3), a (1:12, 4:6), a (1:12, 7:9), a (1:12, 10:12)がそれぞれ、proc (1)、proc (2)、proc (3)、proc (4)に割り付けられる。

【0019】以下、並列化コンパイラ100の中の個々の処理を説明する。構文解析部101は上記ソースプログラム110を入力して辞書120、中間語130を生成する。中間語130はソースプログラム110に対応しているため、以下の説明では、図8のソースプログラム110を、中間語130のソースプログラムイメージの表現として用いる。

【0020】図10は、構文解析部101が生成する辞書120を示したものである。1000は、配列aに対する辞書120である。本説明文ではスカラー変数bとcに対する辞書は説明で用いないため、省略する。1001は変数名を示し、配列名である“a”という文字列が設定されている。1002は変数の型で、実数型を示す“REAL”が設定されている。1003は変数長で、4バイト変数を示す“4”が設定されている。1004は変数が配列か否かを表すフラグで、aは配列なので“TRUE”が設定されている。

【0021】1005は配列を細分化すべきか否かを表すフラグであり、“TRUE”は配列を細分化すべきであることを表す。この値は形状規則数判定104で設定される。1006は変数が配列の場合、その次元数を表す。配列aは2次元配列なので、値2が設定されている。1007は変数が配列の場合、その配列の各次元の寸法宣言情報を格納した配列辞書1040を指すポインタである。1008は細分化情報1100へのポインタである。これは配列細分化105で設定される。1009は高次元化辞書1200へのポインタである。これは配列高次元化106で設定される。1010はデータ分散情報の収集リストの先頭1020へのポインタである。これは形状収集103で設定される。

【0022】1040は配列の各次元の寸法宣言情報を格納した配列辞書であり、辞書1000のフィールド1007から指されている。1041は配列aの第1次元目の下限値と上限値が、各々、1と12であることを表わしている。1042は配列aの第2次元目の下限値と上限値が、各々、1と12であることを表わしている。

【0023】データ分散指示文解析102は、辞書120及び中間語130を入力して、データ分散情報140を出力する。図9は、データ分散情報140を示したも

のである。上記説明した通り、中間語 130 として図 8 のソースプログラム 110 を用いる。また、図 8 における文 802 に対するデータ分散情報は文 804 に対するデータ分散情報と同じなので、ここでは文 804 と文 810 に対するデータ分散情報、及び文 801 に対する論理プロセッサ情報のみを示す。

【0024】940 は PROCESSORS 指示文 801 を解析して得た PROCESSORS 情報テーブルである。941 は論理プロセッサの変数名を表わす文字列 "proc" である。942 は論理プロセッサの配列次元数であり、値 1 は 1 次元配列であることを示す。943 は論理プロセッサの各次元の寸法情報を表わす配列辞書 950 へのポインタである。950 は論理プロセッサの各次元の寸法情報を表わす配列辞書であり、論理プロセッサ proc の第 1 次元目の下限値と上限値が、各々、0 と 3 であることを示している。

【0025】900 は再分散指示文 804 を解析して得た、配列 a に対するデータ分散情報テーブルである。901 は 804 の指示文に対してデータ分散情報が変化する他の変数へのポインタである。今の場合、そのような変数は存在しないので値 NULL が設定されている。902 は配列 a の次元数であり、値 2 が設定されている。903 は各次元ごとのデータ分散情報を表わす次元毎データ分散情報テーブル 920 へのポインタである。904 は論理プロセッサに対する PROCESSORS 情報テーブルへのポインタであり、proc に対する PROCESSORS 情報テーブル 940 を指している。

【0026】920 は配列 a に対する次元毎データ分散情報テーブルである。921 は第 1 次元目の分散形状が block 分割であることを示している。922 は第 1 次元目の block 分割により、連続する 3 個の配列要素が各論理プロセッサに割り当てられることを示している。この数値 3 は、配列辞書 1040 から得られる配列 a の第 1 次元目の寸法「12」を、配列辞書 950 から得られる論理プロセッサ数「4」で割って、小数点以下の端数を切り上げることで得られる。今の場合は「3」となる。この計算方法は前出の "High Performance Fortran Language Specification" に記述されている。923 は第 2 次元目が分散されないことを示している。924 は第 2 次元目の、連続する 12 個の配列要素が各論理プロセッサに割り当てられることを示している。即ち、これは第 2 次元目の全部が各論理プロセッサに割り当てられることを示している。

【0027】910 は再分散指示文 810 を解析して得た、配列 a に対するデータ分散情報テーブルである。911 は再分散指示文 810 に対してデータ分散情報が変化する他の変数へのポインタである。今の場合、そのような変数は存在しないので値 NULL が設定されている。912 は配列 a の次元数であり、値 2 が設定されている。913 は各次元ごとのデータ分散情報を表わす次元毎データ分散情報テーブル 930 へのポインタである。

ータ分散情報テーブル 930 へのポインタである。

【0028】914 は論理プロセッサに対する PROCESSORS 情報テーブルへのポインタであり、proc に対する PROCESSORS 情報テーブル 940 を指している。930 は配列 a に対する次元毎データ分散情報テーブルである。931 は第 1 次元目が分散されないことを示している。932 は第 1 次元目の、連続する 12 個の配列要素が各論理プロセッサに割り当てられることを示している。即ち、これは第 2 次元目の全部が各論理プロセッサに割り当てられることを示している。933 は第 2 次元目の分散形状が block 分割であることを示している。934 は第 2 次元目の block 分割により、連続する 3 個の配列要素が各論理プロセッサに割り当てられることを示している。この数値 3 の計算方法は 922 の場合と同様である。

【0029】データ分散指示文解析部 102 は、上記説明したデータ分散情報テーブル 900 及び 910 を得るものであり、通常の構文解析処理に、辞書の各次元ごとの寸法と論理プロセッサ数から 922 及び 934 を計算する処理を加えただけのものである。詳細なアルゴリズムは省略する。

【0030】次に、形状収集部 103 はデータ分散情報 140 を入力して、変数ごとのデータ分散情報をまとめるための収集リストを作成して、その収集リストの先頭を変数の辞書 120 に接続する。図 10 の 1020 と 1030 はその結果得られるデータ分散情報の収集リストであり、1020 はその内の先頭であり、辞書 1000 のフィールド 1010 から指されている。1021 は次の収集リストへのポインタであり、次の収集リスト 1030 を指している。1022 は形状収集 103 が収集するデータ分散情報 900 へのポインタである。1031 は次の収集リストへのポインタであり、もうリストがないので、値 NULL が設定されている。1032 は形状収集 103 が収集するデータ分散情報 910 へのポインタである。

【0031】形状種別数判定部 104 は、辞書 120 及びデータ分散情報 140 を入力して配列のデータ分散の種別数を判定し、種別数が 2 の時は配列を細分化すべきと判定し、その結果を辞書 120 に反映させる。まず、配列 a に対する辞書 1000 のフィールド 1010 から収集リスト 1020 及び 1030 をたどり、そこから得られるデータ分散情報テーブル 900 及び 910 が各々異なる分散形状 (block, *)、(*, block) であることからデータ分散の種別数 2 を得る。その結果、細分化すべきと判定し、細分化フラグ 1005 の値を TRUE に設定する。

【0032】配列細分化部 105 は、辞書 120 及びデータ分散情報 140 を入力して配列を細分化し、その結果を辞書 120 に反映させる。図 2 は配列細分化部 105 の処理手順を示したものである。以下、配列 a に対す

る処理を示す。まず、形状種別判定部 104 で設定された 1005 のフラグの値が TRUE なので、配列 a は細分化すべきであり、ステップ 200 は Yes となる。次に、ステップ 201 で 1 の値が 1 となる。収集リスト 1020 と 1030 から、配列 a のデータ分散情報テーブル 900、910 をたどり、それから指される 921 及び 931 より、配列 a の 1 次元目のデータ分散形状が “block” と “*” とわかる。今の場合、202 におけるブロック分割の記法における m の値は、922 の値 3 にあたるので、ステップ 202 は Yes となる。

【0033】ステップ 203 では、細分化フラグ TRUE、細分散 block、細分幅 m として 3、細分 P 数として 1 次元目が block 分散となるデータ分散指示文 804 の、block に対する論理プロセッサ数 4 を、細分化情報の 1 次元目に設定する。

【0034】図 11 は細分化情報 1100 を示したものである。1101 は 1 次元目に対する細分化フラグの値を、1102 は 1 次元目に対する細分散を、1103 は 1 次元目に対する細分幅を、1104 は 1 次元目に対する細分 P 数を示している。同様に、1105 は 2 次元目に対する細分化フラグの値を、1106 は 2 次元目に対する細分散を、1107 は 2 次元目に対する細分幅を、1108 は 2 次元目に対する細分 P 数を示す。今、1 が 1 なので、ステップ 203 で設定された情報は細分化情報 1100 の 1 次元目に対するフィールドに格納される。即ち、細分化フラグの値 TRUE が 1101 に、細分散 block が 1102 に、細分幅 3 が 1103 に、細分 P 数 4 が 1104 に格納される。

【0035】次に、ステップ 207 において、1 の値が 2 になる。配列 a は 2 次元なのでステップ 208 は No となり、処理はステップ 202 に戻る。収集リスト 1020 と 1030 から、配列 a のデータ分散情報テーブル 900 及び 910 をたどり、それから指される 923、933 より、配列 a の 2 次元目のデータ分散形状が “*” と “block” とわかる。今の場合、ステップ 202 におけるブロック分割の記法における m の値は、934 の値 3 にあたるので、ステップ 202 は Yes となる。

【0036】ステップ 203 では、細分化フラグ TRUE、細分散 block、細分幅 m として 3、細分 P 数として 2 次元目が block 分散となるデータ分散指示文 810 の、block に対する論理プロセッサ数 4 を、細分化情報の 2 次元目に設定する。今、1 が 2 なので、ステップ 203 で設定された情報は細分化情報 1100 の 2 次元目に対するフィールドに格納される。即ち、細分化フラグの値 TRUE が 1105 に、細分散 block が 1106 に、細分幅 3 が 1107 に、細分 P 数 4 が 1108 に格納される。

【0037】次に、ステップ 207 において、1 の値が 3 になる。配列 a は 2 次元なのでステップ 208 は Yes

s になる。他にデータ分散配列はないので、ステップ 209 は No となり処理を終了する。

【0038】次に、配列高次元化部 106 は、辞書 120 及びデータ分散情報 140 を入力して配列宣言を高次元化することにより配列要素のアドレス順序を変更し、その結果を辞書 120 に反映させる。図 3 は配列高次元化部 106 の処理手順を示したものである。以下、配列 a に対する処理を示す。

【0039】まず、形状種別判定部 104 で設定された 1005 のフラグの値が TRUE なので、配列 a は細分化すべきであり、ステップ 300 は Yes となる。ステップ 301 では、配列 a の辞書 1000 のフィールド 1006 より配列の次元数 K は 2、細分化情報 1100 のフラグ 1101 と 1105 が両方とも TRUE なので、細分化すべき次元数 J は 2 で、K+J=4 となって、4 次元の高次元化辞書の領域を確保する。

【0040】図 12 は高次元化辞書 1200 を示したものである。1201 は高次元化後の配列の次元数、1202 は高次元化後の配列の次元毎情報テーブル 1210 へのポインタ、1210 は次元毎情報テーブル、1211、1214、1217、1220 は各々、高次元化後の配列の 1、2、3、4 次元目の高次元化フラグである。高次元化フラグには、高次元化の対象次元なら “changed” が、高次元化により新しく作成される次元なら “new” が、高次元化の対象でない次元なら “not changed” が設定される。

【0041】1212、1215、1218、1221 は各々、高次元化後の配列の 1、2、3、4 次元目の高次元化対応次元である。高次元化対応次元には、高次元化の対象次元なら、その次元に対応する新しく作成される次元が、高次元化により新しく作成される次元なら、その元の高次元化対象次元が設定され、高次元化の対象でない次元なら 0 が設定される。1213、1216、1219、1222 は各々、高次元化後の配列の 1、2、3、4 次元目の上下限値が設定される。

【0042】次に、ステップ 302 で 1 の値が 1、J の値が 0 となる。細分化情報 1100 の 1 次元目の細分化フラグ 1101 の値が TRUE なので、ステップ 303 は Yes となる。次に、ステップ 304 で J の値を 1 にする。ステップ 305 で高次元化辞書 1200 の第 1 次元目の高次元化フラグ 1211 の値を changed に設定する。辞書 1000 中の 1041 で示された、配列 a の第 1 次元の下限値 1 と上限値 2、細分化情報 1100 のフィールド 1104 が示す細分 P 数の値 4 より、細分化後の下限値 1、細分化後の上限値 $1 + (12/4) - 1 = 3$ を得る。これらから得られる下限値と上限値の組 1:3 を高次元化辞書 1200 の第 1 次元目に対するフィールド 1213 に設定する。また、高次元化対応次元 K+J の値 3 を 1212 に設定する。

【0043】ステップ 306 で、高次元化辞書の K+J

= 3 次元目の高次元化フラグ 1217 の値を new、下限値を 1、上限値を細分化 P E 数の値 4 として、組 1: 4 を 1219 に設定する。また、高次元化対応次元 I の値 1 を 1218 に設定する。ステップ 308 で、I の値が 2 になる。I の値 2 は、K の値 2 より大きいことはないで、ステップ 309 は No となり、ステップ 310 により次の次元に処理を移し、ステップ 303 に戻る。

【0044】細分化情報 1100 の 2 次元目の細分化フラグ 1105 の値が TRUE なので、ステップ 303 は Yes となる。ステップ 304 で J の値は 2 になる。ステップ 305 で高次元化辞書 1200 の第 2 次元目の高次元化フラグ 1214 の値を changed に設定する。辞書 1000 中の 1042 で示された、配列 a の第 2 次元の下限値 1 と上限値 12、細分化情報 1100 のフィールド 1108 が示す細分化 P E 数の値 4 より、細分化後の下限値 1、細分化後の上限値 $1 + (12/4) - 1 = 3$ を得る。これから得られる下限値と上限値の組 1: 3 を高次元化辞書 1200 の第 2 次元目に対するフィールド 1216 に設定する。また、高次元化対応次元 K+J の値 4 を 1215 に設定する。

【0045】ステップ 306 で、高次元化辞書の K+J = 4 次元目の高次元化フラグ 1220 の値を new、下限値を 1、上限値を細分化 P E 数の値 4 として、組 1: 4 を 1222 に設定する。また、高次元化対応次元 I の値 2 を 1221 に設定する。ステップ 308 で、I の値が 3 になる。I の値 3 は、K の値 2 より大きいので、ステップ 309 は Yes となる。データ分散配列は他にないので、ステップ 311 は No となり、配列高次元化の処理を終了する。

【0046】次に、プログラム変換部 107 は、辞書 120、データ分散情報 140 及び中間語 130 を入力してプログラムを変換し、その結果を中間語 130 に反映し、またループテーブル 150 を出力するプログラム変換を行う。プログラム変換部 107 はループ変換部 1071 と添字変換部 1072 からなる。ここでは、中間語 130 として、図 8 の文 803 と文 817 に対応するループ k と、文 805 から文 809 までのループ j、ループ i のみを対象として説明し、文 811 と文 815 に対応するループ j と文 812 から文 814 までに対応するループ i に関する処理は省略する。

【0047】図 4 はループ変換部 1071 の処理手順を説明したものである。プログラム分割解析ステップ 400 では辞書 120、データ分散情報 140 及び中間語 130 を入力して、ループテーブル 150 を出力する。図 13 はループテーブル 150 を示す。1300 は図 8 の文 803 と文 817 に対応するループ k のループテーブル、1310 は文 805 と文 809 に対応するループ j のループテーブル、1330 は文 806 から文 808 までに対応するループ i のループテーブル、1320 は文 811 と文 815 に対応するループ j のループテーブル

を示す。文 812 から文 814 までに対応するループ i のループテーブルは説明に使わないため省略されているが、それは 1320 から接続されている。

【0048】1301 はループ k の一つ内側にあるループ群の先頭ループのループテーブルへのポインタであり、1310 を指している。1302 はループ k の一つ外側にあるループのループテーブルへのポインタであり、ループ k にはそのようなループはないので、値 NULL が設定されている。1303 は当該ループのループ制御変数の辞書へのポインタであり、k を指している。

1304 はループネストであり、一番外側のループなので、1 が設定されている。1305 は当該ループをベ어링するかを示すフラグであり、これはループベ어링解析 401 で設定される。1306 はループの上下限値を示し、下限値 1、上限値 10 が設定されている。1307 はループをプログラム分割した結果、作成されるプログラム分割テーブルの先頭テーブルのポインタである。ループ k はプログラム分割対象ループではないので値 NULL が設定されている。1308 は後続のループのループテーブルへのポインタで、ループ k には後続ループはないので値 NULL が設定されている。

【0049】同様にして、1311 はループ j の内側ループであるループ i のループテーブル 1330 へのポインタ、1312 はループ j の外側ループであるループ k のループテーブル 1300 へのポインタ、1313 はループ制御変数 j の辞書へのポインタ、1314 はループ j のネスト 2、1315 はループ j をベ어링するかを示すフラグ、1316 はループ j の上下限値、1317 はプログラム分割テーブルの先頭テーブル 1340 へのポインタ、1318 は後続ループである、811 と 815 が示すループ j のループテーブルへのポインタを示す。

【0050】同様にして、1331 はループ i の内側ループがないため値 NULL、1332 はループ i の外側ループであるループ j のループテーブル 1310 へのポインタ、1333 はループ i のループ制御変数 i の辞書へのポインタ、1334 はループ i のネスト 3、1335 はループ i をベ어링するかを示すフラグ、1336 はループ i の上下限値、1337 はプログラム分割テーブルの先頭テーブル 1360 へのポインタ、1338 は後続ループがないため値 NULL を示す。ループテーブルの内、以上の内容は中間語 130 から容易に得られる。

【0051】1340 及び 1350 はループ j に対するプログラム分割テーブルである。1341 は本テーブルが有効な論理プロセス番号の範囲を示し、論理プロセス番号 1 に対して有効であることを示す。1342 はプログラム分割後のループの上下限値であり、下限値が 2、上限値が 3 であることを示す。1343 は次のプログラム分割テーブルへのポインタであり、1350 を指

している。1351は本テーブルが有効な論理プロセス番号の範囲を示し、論理プロセス番号2から4までに対して有効であることを示す。1352はプログラム分割後のループの上下限値であり、下限値が1、上限値が3であることを示す。1353は次のプログラム分割テーブルへのポインタであるが、もうないので値NULLが設定されている。上記2つのプログラム分割テーブルにより、ループjは論理プロセス番号1に対しては、ループ上下限値が2と3、論理プロセス番号2から4に対しては、ループ上下限値が1と3、になることがわかる。

【0052】上記プログラム分割テーブルを得るための処理はジーマ・ヒラナダニ、ケン・ケネディ、チャウ・ウェン・ツェン著「コンパイルング フォートラン ディーフォー エムアイエムディ ディストリビューテッド メモリ マシンズ」コミュニケーションズ オブザ エシーエム、ボリューム35、ナンバー8、オーガスト、1992、第66頁から第80頁 (Seena Hirandanani, Ken Kennedy, Chau-Wen Tseng, "Compiling Fortran D for MIND Distributed-Memory Machines", Communications of the ACM, Vol. 35, No. 8, August, 1992, pp.66-80) に詳しい。

【0053】次にループピーリング解析ステップ401は中間語130中の配列参照テーブルとループテーブルを入力し、各々に対してループピーリング後の情報を追加する。

【0054】図14は文807中のa(1, j-1)に対する配列参照テーブルを示したものである。1400はa(i, j-1)に対する配列参照テーブルである。1401は配列aの辞書へのポインタである。1402は配列の各次元の添字情報を表す添字テーブルの先頭へのポインタであり、1410を指している。1403は各次元ごとのピーリング情報を表すテーブルへのポインタであり、1430を指している。これはループピーリング解析ステップ401で設定される。1410はa(i, j-1)の1次元目の添字テーブルである。1411は2次元目に対する添字テーブルへのポインタであり、1420を指している。1412は添字を表す中間語へのポインタであり、iを表す中間語を指している。1420はa(i, j-1)の2次元目の添字テーブルである。1421は配列aには3次元目が存在しないので、値NULLが設定されている。1422は添字を表す中間語へのポインタであり、j-1を表す中間語を指している。1430及び1440は次のループピーリング解析で作成されるテーブルであり、その処理の中で説明する。

【0055】図5はループピーリング解析ステップ401の詳細な処理手順を示したものである。ここでは図14で示された配列参照テーブル1400に対する処理のみ説明する。ステップ500でlの値は1になる。配列

aの1次元目は細分化情報1100中のフィールド1101がTRUEなので細分化する次元であり、ステップ501はYesとなる。

【0056】1412より当次元に現れる変数はiであり、これはループテーブル1330よりループ制御変数であることがわかる。この1に分散後ループ範囲1362を代入して、当次元の添字範囲は1:3になる。一方、データ分散後の当次元の添字範囲は高次元化辞書1200のフィールド1213より1:3であるから、分散後ループ範囲から得られた添字範囲1:3からデータ分散後の添字範囲1:3をひいた残りの範囲は空集合となり、ステップ502の結果は空集合である。よって、ステップ503はYesとなる。

【0057】ステップ505で当次元はピーリングしない、とし、ピーリングすべきループ範囲とピーリングすべき添字のはみ出す方向を各々、0とする。次に、ステップ506で、ピーリング情報テーブル1430を作成する。ここで、1431は次のピーリング情報テーブルへのポインタ、1432はピーリングするかを示すフラグ、1433は当次元に現れるループ制御変数、1434はピーリングすべきループ範囲、1435はピーリングすべき添字のはみ出す方向、1436は当ループ制御変数を持つループテーブルへのポインタである。

【0058】よって、今の場合、1431には、まずNULLが設定される。但し、このフィールドは次の2次元目の処理で、新規に作成されるピーリング情報テーブル1440を指すように変更される。1432にはピーリングしないのでFALSEが、1433には当次元に現れるループ制御変数iが、1434には0が、1435にも0が、1436にはループiに対するループテーブル1330へのポインタが設定される。

【0059】次に、ステップ507では、現在は1次元目の処理をしているので、Yesとなる。ステップ508で、lの値は2になる。配列aの2次元目は細分化情報1100中のフィールド1105がTRUEなので細分化する次元であり、ステップ501はYesとなる。1422より当次元に現れる変数はjであり、これはループテーブル1310よりループ制御変数であることがわかる。このjに分散後ループ範囲1342、1352を代入し、1341、1351の論理プロセス番号範囲も含めると、当次元の添字範囲は、論理プロセス番号範囲が1:1の時は1:2、論理プロセス番号範囲が2:4の時は0:2となる。

【0060】一方、データ分散後の当次元の添字範囲は高次元化辞書1200のフィールド1216より1:3であるから、分散後ループ範囲から得られた添字範囲からデータ分散後の添字範囲をひいた残りの範囲は論理プロセス番号範囲が2:4の時に0:0となり、ステップ502の結果は0:0である。よって、ステップ503はNoとなる。

【0061】ステップ504では、まず、「当次元はピーリングする」とする。次に、0:0が2次元目の添字である「j-1」と等しくなることより、「j」の値は1:1となる。よって、はみ出す時のループ範囲であるピーリング範囲は1:1である。また、このピーリング範囲は論理プロセッサ番号範囲が2:4の時に生じるので、ループテーブル1310の分散ループ範囲のうち、この論理プロセッサ番号範囲に対応するテーブル1350を細分する。即ち、テーブル1350を、ピーリング範囲1:1の範囲を持つテーブルとそれ以外の範囲からなる

テーブルに分割する。
【0062】図15は細分化後のループ分散範囲テーブルである。テーブル1350のループ分散範囲は1:3から1:1に変更されている。新たにテーブル1500が作成され、1353は1500を指す。1501は細分されるループ分散範囲の論理プロセッサ番号範囲2:4に設定される。1502は元のループ分散範囲は1:3から1:1を引いた範囲2:3に設定される。1503は他に指すべきテーブルがないので、値NULLが設定される。

【0063】次に、ステップ506より、ピーリング情報テーブル1440を作成する。まず、1441には次のテーブルがないのでNULLが設定され、1431には1440へのポインタが設定される。1442には、ピーリングすることを示すフラグTRUEが、1443には当次元に現れるループ制御変数である「j」が、1444にはステップ504で計算した1:1を設定する。ループ分散後のループ範囲1342及び1352の内、1:1は1352の下限を含むので、はみ出す方向として下限方向を示す-1を、1445に設定する。1446には、ループ「j」に対するループテーブル1310へのポインタを設定する。

【0064】次に、配列aにはもう処理すべき次元がないので、ステップ507はNoとなる。上記説明したように、図5では図14の配列参照に対する処理だけを説明するので、ステップ509はNoになり、これでループピーリング処理の説明を終了する。

【0065】ここで、図4に戻り、ループ変換の説明を続ける。ループ変換の残りの処理では、図13のループテーブル群から図16及び17で示されたループテーブル群を生成し、図18のプログラムを生成する。まず、ステップ402で、最初の最内側ループであるループiを検出する。以降の処理で、処理済みの分散ループに対しては、ループ分散範囲テーブルへのポインタをNULLにする。よって、ステップ403で、未処理の分散ループであることは、分散ループであることを示すフラグがTRUEで、かつ、ループ分散範囲テーブルへのポインタがNULLでないことからわかる。図13の、ループiに対するループテーブル1330で、1335はTRUEで、1337はNULLでないので、ループiは

未処理の分散ループであり、ステップ403はYesである。

【0066】ステップ404では、1330の一つ内側にループテーブルを作成する。図16はステップ404により現ループテーブル1330(LT1)の一つ内側に新たにループテーブル1600(LT2)が作成された様子を表わした図である。

【0067】以下、ステップ404に従って図16を説明する。まず、ループ分散テーブル1360は1個だけなので、LT1以下のループテーブルのコピーは作成する必要はない。LT1に対するループ制御変数名をi2とし、1333に設定する。LT1のループ範囲1336をループ分散範囲テーブル1360の論理プロセッサ番号範囲1361である1:4に設定する。

【0068】ループテーブル1600は一つ内側のループに対するテーブルなので、一つ内側のループ先頭を指すポインタ1331を1600を指すように設定する。一つ外側のループテーブルへのポインタ1331、ループネスト1334、分散ループフラグ1335、後続ループへのポインタ1338はそのままである。ループ分散範囲テーブル1360へのポインタ1337をNULLに設定して、このループテーブルが処理済みであることを示す。

【0069】次に、LT2に対するループ制御変数名をi1とし、1603に設定する。LT2のループ範囲1606をループ分散範囲テーブル1360のループ分散範囲1362である1:3に設定する。ループテーブル1600は最内側ループなので、1601はNULLに、一つ外側のループテーブルへのポインタである1602は1330を指すように設定される。また、1604、1605、1607、1608は各々、O、TRUE、NULL、NULLに初期化される。1605がTRUEで1607がNULLなので、ループテーブル1600は処理済みであることを示す。

【0070】次に、1338はNULLなので後続ループはなく、ステップ405はNoである。1332は外側ループテーブル1310を指しているのので、ステップ407はTRUE(Yes)であり、ステップ408で処理は外側ループテーブル1310に移り、ステップ403へ戻る。1315はTRUEだが、1317はNULLでないので、ループテーブル1310は未処理の分散ループであり、ステップ403はYesである。

【0071】ループテーブル1310のループ分散範囲テーブルの個数は図15より3個なので、ステップ404では、ループテーブル1310(LT1)の一つ内側にループテーブル(LT2)を作成し、LT1以下のループテーブルのコピーを2個作成して、LT1の後続ループとして接続する。

【0072】図17はこれらのテーブルが作成された様子を表わした図である。1700がLT2を、171

0, 1720, 1730, 1740がLT1以下のループテーブルの第1のコピーを、1750, 1760, 1770, 1780がLT1以下のループテーブルの第2のコピーを表わす。

【0073】以下、ステップ404に従って図17を説明する。まず、LT1とそのコピーに対するループ制御変数名をj2とし、1313, 1713, 1753に設定する。LT1とそのコピーのループ範囲1316, 1716, 1756をループ分散範囲テーブル1340, 1350, 1500の論理プロセス番号範囲1341, 1351, 1501である1:1, 2:4, 2:4に設定する。ループテーブル1700, 1720, 1760は、各々、ループテーブル1310, 1710, 1750の一つ内側のループに対するテーブルなので、一つ内側のループ先頭を指すポインタ1311, 1711, 1751を、各々、1700, 1720, 1760を指すように設定する。一つ外側のループテーブルへのポインタ1312はそのままであり、1712, 1752はNULLに初期化される。

【0074】ループネスト1314はそのままであり、1714, 1754は0に初期化される。これらの値は後のループネスト決定ステップ409で、括弧内の値に設定される。分散ループフラグ1315はそのままであり、1715, 1755はTRUEに設定される。ループ分散範囲テーブル1340へのポインタ1317、及び1717, 1757をNULLに設定して、これらのループテーブルが処理済みであることを示す。後続ループテーブルへのポインタは、1320を指していた1318は1710を、1718は1750を、1758は1320を指すように設定される。

【0075】次に、LT2とそのコピーに対するループ制御変数名をj1とし、1703, 1723, 1763に設定する。LT2とそのコピーのループ範囲1706, 1726, 1766をループ分散範囲テーブル1340, 1350, 1500のループ分散範囲1342, 1352, 1502である2:3, 1:1, 2:3に設定する。

【0076】ループテーブル1330, 1730, 1770は、各々、ループテーブル1700, 1720, 1760の一つ内側のループに対するテーブルなので、一つ内側のループ先頭を指すポインタ1701, 1721, 1761を、各々、1330, 1730, 1770を指すように設定する。ループテーブル1310, 1710, 1750は、各々、ループテーブル1700, 1720, 1760の一つ外側のループに対するテーブルなので、一つ外側のループ先頭を指すポインタ1702, 1722, 1762を、各々、1310, 1710, 1750を指すように設定する。ループネスト1704, 1724, 1764は0に初期化される。これらの値は後のループネスト決定409で、括弧内の値に設

定される。分散ループフラグ1705, 1725, 1765はTRUEに設定される。ループ分散範囲テーブルへのポインタ1707, 1727, 1767をNULLに設定して、これらのループテーブルが処理済みであることを示す。後続ループテーブルへのポインタは1708, 1728, 1768はNULLに初期化される。

【0077】1730と1770, 1740と1780は各々、1330, 1600のコピーであり、外側ループ、内側ループを指すポインタ、ループネスト以外は全て同じ値となる。即ち、1733, 1773とは1333と同じ値で、1735から1738まで、1775から1778までは1335から1338までと同じ値である。また、1743, 1783は1603と同じ値で、1745から1748までと1785から1788までは1605から1608までと同じ値である。ループネストである1734, 1774, 1744, 1784は0に初期化される。一つ内側のループテーブルを指すポインタは、1731が1740を、1741はNULLを、1771は1780を、1781はNULLを指すように設定される。一つ外側のループテーブルを指すポインタは、1732が1720を、1742は1730を、1772は1760を、1782は1770を指すように設定される。

【0078】次に、ループテーブル1310の元々の後続テーブルは1320であるが、本実施例では1320以降のテーブルに対する処理は説明しないので、ステップ405はNoとなる。1312より、ループテーブル1310の外側ループテーブル1300が存在するので、ステップ407はYesとなり、ステップ408で処理をループテーブル1300に移し、ステップ403へ戻る。

【0079】1305がFALSEなので、ループテーブル1300に対応するループkは分散ループでなく、ステップ403はNoとなる。1308はNULLなので、後続ループはなく、ステップ405はNoとなる。1302はNULLなので、外側ループはなく、ステップ407はNoとなる。

【0080】次に、ループネスト決定ステップ409で、図17のループテーブルのネストを決定する。この処理は最も外側のループから順番に決定する簡単な処理なので、詳細は省略する。図17の、1304, 1314, 及び、1704, 1334, 1604, 1714, 1724, 1734, 1744, 1754, 1764, 1774, 1784の括弧内の数値が決定されたループネストである。

【0081】次にループ多重化・ループピーリングステップ410で図17のループテーブル群に従って、ループを表わす元の中間語に、ループ多重化とループピーリングを適用する。図18は、元のループ中間語である図8の文803から文817に、ループ多重化とループピ

ーリングを適用した結果のプログラムである。

【0082】文1810から文1835までが図8の文805から文809に対応する部分である。このうち、文1810から文1815までがループテーブル1310、1700、1330、1600に対応し、文1820から文1825までがループテーブル1710、1720、1730、1740に対応し、文1830から文1835までがループテーブル1750、1760、1770、1780に対応する。同様にして、文1840から文1880までが図8の文811から文815に対応する部分である。

【0083】図4に戻り、最後に後続ループネストはないので、ステップ411はNoとなり、ループ変換部1071の処理を終了する。

【0084】次に、図1の添字変更部1072の処理を行う。図6は添字変更部1072の処理手順を説明した図である。ここでは、文1814と文1824中の配列参照a(i, j-1)に対する処理を、その配列参照テーブルを示した図14、処理結果得られるプログラムを示した図19を用いて説明する。

【0085】まず、ステップ600でIの値は1となり、1次元目の処理を行う。細分化情報テーブル中の1次元目の細分化フラグ1101の値がTRUEなので、この次元は細分化する次元であり、ステップ601はYesとなる。高次元化辞書1200の1次元目に対する高次元化対応次元1212は3となっているので、ステップ602では、配列参照に3次元目を追加する。文1814はループ中にあるので、ステップ603はYesである。

【0086】ステップ604で、現在の配列参照テーブル1400の1次元目の添字は1411より1で、ループ制御変数Iを持っていたループテーブルへのポインタは1436より1330とわかるので、1次元目の添字を値が先に増加する内側ループのループ制御変数である1603のi1に、3次元目の添字を外側ループのループ制御変数である1333のi2に設定する。現配列参照の1次元目に対するピーリングフラグ1432はFALSEなので、ステップ606はNoとなる。配列aは元々2次元なので、ステップ608はYesとなり、ステップ610でIの値を2にして処理を2次元目に移し、ステップ601に戻る。

【0087】細分化情報テーブル中の2次元目の細分化フラグ1105の値がTRUEなので、この次元は細分化する次元であり、ステップ601はYesとなる。高次元化辞書1210の2次元目に対する高次元化対応次元1215は4となっているので、ステップ602では、配列参照に4次元目を追加する。文1814はループ中にあるので、ステップ603はYesである。

【0088】ステップ604で、現在の配列参照テーブル1400の2次元目の添字は1422よりj-1で、

ループ制御変数Jを持っていたループテーブルへのポインタは1446より1310とわかるので、2次元目の添字を値が先に増加する内側ループのループ制御変数である1703のj1を用いてj1-1に、4次元目の添字を外側ループのループ制御変数である1313のj2に設定する。

【0089】現配列参照の2次元目に対するピーリングフラグ1442はTRUEだが、現配列参照のピーリングすべきループ範囲は1444より1:1である。一方、現配列参照を含み、元々ピーリングループ制御変数Jを持っていたループのループテーブル1310の、一つ内側ループのループ範囲1706は2:3であるから、1:1は2:3に含まれておらず、ステップ606はNoとなる。これは、ピーリングに伴って、jに対する添字変更の特別処理が必要なループ範囲が1:1であるが、現在のループ範囲はその範囲を含んでいないので、特別な処理が不要であることを示している。

【0090】配列aの次元はもうないので、ステップ608はNoとなる。以上で、文1814中の配列参照a(i, j-1)は、文1915中の配列参照a(i1, j1-1, i2, j2)に変えられ、配列参照はまだあるので、ステップ609はYesとなり、ステップ611で次の配列参照である文1824のa(i, j-1)に処理を移す。

【0091】この配列参照に対して、1次元目の処理は文1814の場合と同じなので、省略して、2次元目の場合のみ説明する。ステップ601、602、603、604の処理は文1814の配列参照の場合と同様で、その結果、文1824のa(i, j-1)は、一旦、文1915の配列参照a(i1, j1-1, i2, j2)のように変更される。

【0092】次に、ピーリングフラグ1442はTRUEで、現配列参照のピーリングすべきループ範囲は1444より1:1である。一方、現配列参照を含み、元々ピーリングループ制御変数Jを持っていたループのループテーブル1310のコピーになるループテーブル1710であり、その一つ内側ループのループ範囲1726は1:1である。よって、両者は一致し、ステップ606はYesとなる。ピーリング添字のみ出す方向は1445より-1、現次元の寸法は1216より3なので、j1はj1+3、j2はj2-1となる。これより、j1-1はj1+2に、j2はj2-1になり、文1925の配列参照a(i1, j1+2, i2, j2-1)が得られる。

【0093】次に、配列aは2次元配列なので、ステップ608はNoとなる。本説明では2つの配列参照についての説明するので、ステップ609はNoとなり、添字変更部1072の処理を終了する。他の配列参照についても同様に処理することで、図19の中間語が得られる。以上より、プログラム変換部107の処理を終

了する。

【0094】次に、プログラム並列化部108の処理を説明する。図7はプログラム並列化部108の処理手順を説明した図である。データ分散情報変更ステップ700は、データ分散情報140と高次元化辞書1200を入力して、変更したデータ分散情報を出力する。図20は変更後のデータ分散情報である。データ分散情報変更の処理は容易なので、概略を記す。

【0095】配列高次元化部106の処理によって、配列の次元が1201に示されるように4になったので、図20における902、912は図9における2から4になる。これに伴い、2000、2010には4次元分のテーブルを確保する。次に、高次元化辞書において、フラグの値が“changed”で、かつ、データ分散が指定された次元の分散形状は、次元毎分散情報テーブルの該当する次元では“*”と変更され、上記次元に対応する、フラグの値が“new”である次元の分散形状は、上記“changed”フラグを持つ次元の、元の分散形状に設定される。但し、ブロック分割の幅は再計算される。その他の次元は、“*”と設定される。

【0096】まず、図9のデータ分散情報テーブル900について処理する。高次元化辞書の第1次元目は、フラグ1211の値が“changed”で、データ分散情報テーブル900に対応する第1次元目のデータ分散形状921はblockなので、この次元の分散形状は“*”に変更される。2001は“*”となり、2002には1213より、高次元化辞書の第1次元目の寸法3が設定される。上記第1次元目に対応する次元は1212より3次元目であり、3次元目のフラグ1217の値が“new”なので、この次元はデータ分散するように変更される。よって、2005はblockとなり、この次元の上下限1219よりわかる寸法4を、950よりわかる論理プロセス数4で割った値1がブロック分割の幅となるので、この値1を2006に設定する。

【0097】一方、第2次元目は、フラグ1214の値は“changed”であるが、データ分散情報テーブル900に対応する第2次元目のデータ分散形状923は“*”なので、この次元の分散形状は“*”に設定される。よって、2003は“*”となり、2004には1216より、高次元化辞書の第2次元目の寸法3が設定される。また、上記2次元目に対応する次元は1215より4となり、第4次元目の分散形状も“*”に設定される。よって、2007は“*”となり、2008には1222より、高次元化辞書の第4次元目の寸法4が設定される。

【0098】次に、図9のデータ分散情報テーブル910について処理する。高次元化辞書の第1次元目は、フラグ1211の値は“changed”であるが、データ分散情報テーブル910に対応する第1次元目のデータ分散形状931は“*”なので、この次元はデータ分

散しないように設定される。よって、2011は“*”となり、2012には1213より、高次元化辞書の第1次元目の寸法3が設定される。また、上記第1次元目に対応する次元は1212より3となり、第3次元目もデータ分散されないように設定される。よって、2015は“*”となり、2016には1219より、高次元化辞書の第3次元目の寸法4が設定される。

【0099】一方、高次元化辞書の第2次元目は、フラグ1210の値が“changed”で、データ分散情報テーブル910に対応する第2次元目のデータ分散形状933はblockなので、この次元はデータ分散しないように変更される。2013は“*”となり、2014には1216より、高次元化辞書の第2次元目の寸法3が設定される。上記第2次元目に対応する次元は1215より4次元目であり、4次元目のフラグ1220の値が“new”なので、この次元はデータ分散するように変更される。よって、2017はblockとなり、この次元の上下限1222よりわかる寸法4を、950よりわかる論理プロセス数4で割った値1がブロック分割の幅となるので、この値1を2018に設定する。

【0100】次に、図20のデータ分散情報に基づいて、プログラム分割解析ステップ701、通信解析ステップ702を実行する。これらに対する処理方法はジャー・ヒランダン、ケン・ケネディ、チャウ・ウェン・ツェン著「コンパイルリング フォートラン ディーフォーエムアイエムディー ディストリビューテッドメモリー シーンズ」コミュニケーションズ オブザエーション・エム、ボリューム35、ナンバー8、オグアスト、1992、第6頁から第80頁 (Seena Hiranandani, Ken Kennedy, Chau-Wen Tseng, "Compiling Fortran D for MIMD Distributed-Memory Machines", Communications of the ACM, Vol. 35, No. 8, August, 1992, pp.66-80) に書かれている。また、通信生成707、プログラム分割コード生成708、分割データ宣言作成709についても同文獻に書かれている。

【0101】図21はプログラム並列化の処理結果のプログラムを示す図である。以下、図10、図20、図21を用い、処理手順の概略を説明する。配列aは細分化フラグ1005がTRUEなので、ステップ703はYesとなる。通信生成ステップ704では、通常のsend/receive型の論理プロセス間通信の生成を行い、再分散処理ではステップ7041の非バッファリング再分散通信生成により、バッファリングしない再分散通信を生成する。これは図21の文2102及び文2140に対応する。2101の再分散ルーチン“non_buf_remap”は、配列aを、ディスクリプトD1に記述されたデータ分散形状から、ディスクリプトD2に記述されたデータ分散形状に、バッファを使わずにデータ再分散するルーチンである。

25

【0102】文2102のD1は、この時点における配列aのデータ分散形状を記述したディスクリプタである。これは、ループkの1回目の繰り返しでは、プログラム先頭におけるデータ分散であり、文802と文804が元々同じデータ分散であることから、文804でのデータ分散に対応する、図20のテーブル900及び2000が示すデータ分散となる。また、ループkの2回目以降の繰り返しでは、文810でのデータ分散に対応する、図20のテーブル910及び2010が示すデータ分散となる。

【0103】文2102のD2は、再分散ルーチン“non_buf_remap”により、配列aが再分散された後のデータ分散形状を記述したディスクリプタであり、図20のテーブル900及び2000が示すデータ分散情報に対応している。文2140の再分散ルーチン“non_buf_remap”は、配列aを、ディスクリプタD2に記述されたデータ分散形状から、ディスクリプタD1に記述されたデータ分散形状に、バッファを使わずにデータ再分散するルーチンである。

【0104】文2140のD2は、この時点における配列aのデータ分散形状を記述したディスクリプタであり、図20のテーブル900及び2000が示すデータ分散情報に対応している。文2140のD1は、この時点における配列aのデータ分散形状を記述したディスクリプタであり、図20のテーブル910及び2010が示すデータ分散情報に対応している。

【0105】プログラム分割コード生成ステップ705では、ループインデックス生成ステップ7051とグローバル添字生成ステップ7052の処理を実行する。ループインデックス生成ステップ7051では、データ分散する配列次元の添字に含まれるループ制御変数に対応するループのループ範囲を分散して、分散後のループの上下限値を生成する。

【0106】本発明の配列高次元化部106の処理を適用した後の配列では、データ分散情報変更ステップ700によりデータ分散する次元は元のデータ分散次元の高次元化対応次元であり、その次元の寸法は配列高次元化部106の処理および論理プロセス数と等しい。ループ変換部1071により上記データ分散される次元に現れるループ制御変数を持つループのループ長はループ分散範囲テーブルの論理プロセス番号範囲に等しく、図8に対する中間語を入力とした場合、それは論理プロセス数に等しいのでループは解消される。

【0107】グローバル添字生成ステップ7052では、データ分散した次元の添字として、データ分散前の該当する配列要素を表わす添字を用いる。本発明では、データ分散する配列次元の添字はデータ分散前は、論理プロセス番号の内のいずれかと等しくなっているので、データ分散後の添字と同じ論理プロセス番号にする。

26

【0108】文1910から文1936までのデータ分散は、データ分散情報テーブル900と2000で示されたものであり、配列aの第3次元目がブロック分割される。aの第3次元目にあるループ制御変数は1914, 1915, 1924, 1925, 1934, 1935の配列参照においてi2であるので、ループi2は解消され、配列aの第3次元目の添字は論理プロセス番号になる。図21の文2113, 2114, 2123, 2124, 2133, 2134における配列aの第3次元目の添字“my p e”が論理プロセス番号を表わしており、図21の文2110から2135には、ループi2がないことがわかる。

【0109】同様にして、文1940から文1980までのデータ分散は、データ分散情報テーブル910と2010で示されたものであり、配列aの第4次元目がブロック分割される。aの第4次元目にあるループ制御変数は文1952, 1953, 1962, 1963, 1972, 1973の配列参照においてj2であるので、ループj2は解消され、配列aの第4次元目の添字は論理プロセス番号になる。図21の文2152, 2153, 2162, 2163, 2172, 2173における配列aの第4次元目の添字“my p e”が論理プロセス番号を表わしており、図21の文2141から文2180には、ループj2がないことがわかる。

【0110】次にグローバルデータ宣言作成ステップ706では、データ分散配列に対して、高次元化後で、データ分散前の配列宣言を行なう。文2100の宣言がそれに該当する。最後に、コード生成部109は、図21に対応する中間語を入力し、図21のソースプログラムまたはそれをオブジェクトプログラムの形にした並列化プログラム160を出力する。これで第1の実施例の処理フローの説明を終了する。

【0111】図22は本願のデータ再配置やデータ分散を適用する前の配列を用いて、データ再分散の様子を説明したものである。2200は1時間目および2次元目の要素数が各々12の2次元配列aを表わし、縦方向は配列の1次元目を表わす。この次元はFORTRANではアドレスの連続する次元にあたる。また、横方向は配列の2次元目を表わす。2200中の16個の矩形は各々、3x3の部分配列を表わす。

【0112】各部分配列中の矢印の左側の数字は、配列aが(block, *)という方法で4台の論理プロセッサにブロック分散された時の、各々の部分配列が割り付けられる論理プロセス番号を、矢印の右側の数字は、配列aが(*, block)という方法で4台の論理プロセッサにブロック分散された時の、各々の部分配列が割り付けられる論理プロセス番号を表わす。したがって、各部分配列中の矢印とその両側の数字により、その部分配列が(block, *)から(*, block)へのデータ再分散によって、左側の番号の論理プロ

50

27

セッサから右側の番号の論理プロセッサへ送信されることを表す。

【0113】図 23 は従来技術による並列化プログラムである。文 2300 における配列 a の宣言が示すように、配列は第 1 次元目が分割されて寸法が 3 になっている。文 2302 と文 2320 はバッファリングをするデータ再分散ルーチンである。文 2302 の再分散ルーチン "remap" は、配列 a を、ディスクリプタ E1 に記述されたデータ分散形状から、ディスクリプタ E2 に記述されたデータ分散形状に、データ再分散するルーチン 10 である。

【0114】文 2302 の E1 は、この時点における配列 a のデータ分散形状を記述したディスクリプタである。これは、ループ k の 1 回目の繰り返しでは、プログラム先頭におけるデータ分散形状であり、文 802 と文 804 が同じデータ分散であることから、文 804 のデータ分散を表す。図 9 のテーブル 900 及び 920 が示すデータ分散となる。また、ループ k の 2 回目以降の繰り返しでは、文 810 でのデータ分散を表す。図 9 のテーブル 910 及び 930 が示すデータ分散となる。20

【0115】文 2302 の E2 は、再分散ルーチン "remap" により、配列 a が再分散された後のデータ分散形状を記述したディスクリプタであり、図 9 のテーブル 900 及び 920 が示すデータ分散情報に対応している。文 2320 の再分散ルーチン "remap" は、配列 a を、ディスクリプタ E2 に記述されたデータ分散形状から、ディスクリプタ E1 に記述されたデータ分散形状に、データ再分散するルーチンである。文 2320 の E2 は、この時点における配列 a のデータ分散形状を記述したディスクリプタであり、図 9 のテーブル 900 及び 920 が示すデータ分散情報に対応している。文 2320 の E1 は、再分散ルーチン "remap" により、配列 a が再分散された後のデータ分散形状を記述したディスクリプタであり、図 9 のテーブル 910 及び 930 が示すデータ分散情報に対応している。30

【0116】図 24 は従来の並列化によるデータ再分散の様子を、特に、論理プロセッサ 1 (以下、P1 と略す) から論理プロセッサ 2 (以下、P2 と略す) へ送信されるデータに注目して表示したものである。2401 で示される 3 行 12 列の部分配列は、配列 a を (block, *) というデータ分散方法で従来の並列化方法により並列化した時、P1 に割り付けられる分割データである。2400 は分割データ 2401 の元の配列における相対的な位置を示すために、参考までに表示した 12 行 12 列の元の配列である。配列 a は (block, *) でデータ分散されるため、配列の 1 次元目が 4 分割され、その内の一つである 2401 が P1 に割り付けられるデータとなる。2401 中の数字は、P1 に割り付けられたデータ 2401 の配列要素がアドレス順に並ぶ様子 40 を示している。

28

【0117】2411 で示される 12 行 3 列の部分配列は、配列 a を (*, block) というデータ分散方法で従来の並列化方法により並列化した時、P2 に割り付けられる分割データである。2410 は分割データ 2411 の元の配列における相対的な位置を示すために、参考までに表示した 12 行 12 列の元の配列である。配列 a は (*, block) でデータ分散されるため、配列の 2 次元目が 4 分割され、その内の一つである 2411 が P2 に割り付けられるデータとなる。2411 中の数字は、P2 に割り付けられたデータ 2411 の配列要素がアドレス順に並ぶ様子を 40 示している。

【0118】2401 中の網掛け部分 2402 は、(block, *) から (*, block) へのデータ再分散により、P1 から P2 へ送信されるデータを表す。図中に示されたアドレス順により、このデータは 19 番目から 27 番目までの配列要素からなる連続データであることがわかる。一方、2411 中の網掛け部分 2412 は、(block, *) から (*, block) へのデータ再分散により、P2 が P1 から受信するデータの格納先を表す。図中に示されたアドレス順により、このデータは 4 番目から 6 番目、16 番目から 18 番目、28 番目から 30 番目までの配列要素からなる非連続データであることがわかる。

【0119】従って、送信側データは連続であるが、受信側では非連続な位置にデータを格納することが必要である。データ通信ルーチンの中には、送信側データは非連続なものを許すものもあるが、受信側は通常、連続になっていないといけない。この場合、1 回のデータ送信で済む場合には、受信側でどうしてもバッファリングが必要になることがわかる。即ち、送信側データ 2402 は送信ルーチンにより、P2 内のアドレスが連続するバッファに送信され、受信した P2 は、そのバッファ内のデータを 2412 へ、各列毎に 1 回、少なくとも 3 回コピーを行う。

【0120】図 25 は、本発明の並列化によるデータ再分散の様子を、P1 から P2 へ送信されるデータに注目して表示したものである。2500 で示される 12 行 12 列の配列は、配列 a を (block, *) というデータ分散方法で分割して、本発明の並列化方法により並列化した時、P1 に割り付けられるデータである。この配列の大きさはデータ分散前の配列の大きさと同じである。2500 中の数字は、P1 に割り付けられたデータ配列 2500 の配列要素がアドレス順に並ぶ様子を 40 示している。左上から始まる 12 行 3 列の部分、最初の 36 要素のアドレス順が示されている。これにより、太線で区切られた 3 行 3 列の部分は連続アドレスであることがわかる。数字が表示されていない他の部分についても同様である。

【0121】この、本発明による配列要素のアドレスの変更は、一般に、配列 a を 2 次元配列 a (N, N)、P

ロセッサ数を P 、 N は P で割り切れる、とする時、次の *

$$(i, j) \rightarrow ((i-1) \bmod L+1, (j-1) \bmod L+1,$$

ここで、矢印の左辺の (i, j) は元の配列 a の第 i 行第 j 列の配列要素を表し、矢印の右辺は本発明による配列再配置を適用した後の 4 次元配列における対応する配列要素を表す。また、 L は N/P を、 $[]$ は切り捨てを表すガウスの記号である。

【0122】この 4 次元配列の最初の 2 次元から作られる 2 次元配列は、データ配列 2500 中の、太線で区切※

$$(i, j) \rightarrow ((i-1) \bmod 3+1, (j-1) \bmod 3+1, [(i-1)/3]+1, [(j-1)/3]+1) \quad (2)$$

ここで、データ配列 2500 の左上角にある配列要素

(1, 1) は、式 (2) より、(1, 1, 1, 1) に移される。これは、位置配列の配列要素が (1, 1)、即ち、データ配列 2500 の左上角の太線で区切られた 3 行 3 列の部分であり、その 3 行 3 列の部分からなる 2 次元配列中の配列要素が (1, 1) であることを示しており、元の (1, 1) と同じアドレスを指していることがわかる。

【0124】また、データ配列 2500 中のアドレス番号が 83 の部分は、元の配列における配列要素が (5, 7) であり、これは式 (2) により、(2, 1, 2, 3) に移される。これは、位置配列の配列要素が (2, 3)、即ち、2502 を示しており、その位置配列要素が指す、3 行 3 列の部分からなる 2 次元配列中の配列要素が (2, 1) であることを示しており、アドレス番号が 83 の部分を指していることがわかる。

【0125】2501 で示される、アドレス順で、10 番目から 18 番目、46 番目から 54 番目、82 番目から 90 番目、118 番目から 126 番目の、3 行 12 列の部分配列は、配列 a を (block, *) というデータ分散方法で分割して、本発明の並列化方法により並列化した時、 P がその配列要素の計算を担当するデータである。即ち、2502 の配列要素の値の更新は、原則的に P が行なう。2510 で示される 12 行 12 列の配列は、配列 a を (block) というデータ分散方法で分割して、本発明の並列化方法により並列化した時、 P 2 に割り付けられるデータである。この配列の大きさはデータ分散前の配列の大きさと同じである。

【0126】2510 中の一部の数字も同様、 P 2 に割り付けられたデータ 2510 の配列要素がアドレス順に並ぶ様子を示している。2511 で示される、アドレス順で、73 番目から 108 番目の、12 行 3 列の部分配列は、配列 a を (block) というデータ分散方法で分割して、本発明の並列化方法により並列化した時、 P 2 がその配列要素の計算を担当するデータである。即ち、2511 の配列要素の値の更新は、原則的に P 2 が行なう。

【0127】2501 中の網掛け部分 2502 は、(block, *) から (block) へのデータ再分散により、 P 1 から P 2 へ送信されるデータを表す。

* 式 (1) によって表現される。

$$[(i-1)/L]+1, [(j-1)/L]+1) \quad (1)$$

※られた 3 行 3 列の部分に対応し、最後の 2 次元から作られる 2 次元配列は、この太線で区切られた 3 行 3 列の部分の一つの配列要素とみなして得られる 4 行 4 列の位置配列を表す。

【0123】例として、図 25 と同様に、 $N=12$ 、 $P=4$ だとすると、 $L=N/P=3$ となり、式 (1) は次の式となる。

$$(i, j) \rightarrow ((i-1) \bmod 3+1, (j-1) \bmod 3+1, [(i-1)/3]+1, [(j-1)/3]+1) \quad (2)$$

図中に示されたアドレス順により、このデータは 82 番目から 90 番目までの配列要素からなる連続データであることがわかる。一方、2511 中の網掛け部分 2512 は、(block, *) から (block) へのデータ再分散により、 P 2 が P 1 から受信するデータを表す。図中に示されたアドレス順により、このデータはやはり、82 番目から 90 番目までの配列要素からなる連続データであることがわかる。

【0128】従って、送信側データも受信側データも連続となり、データ通信に際してはバッファを介する必要がないことがわかる。これは、配列からバッファへのコピーが不要であることを意味している。以上により、従来技術では、データ再分散に際し、データとバッファ間のコピーが必要であったが、本願により、それが不要になることがわかる。

【0129】尚、本実施例では、分散メモリマシン向けの並列化プログラムに関するデータ再分散の説明を行ったが、分散共有メモリマシンと呼ばれる、メモリは物理的に分散しているが、ハードウェアや OS などにより、ユーザ側には共有メモリに見えるマシン向けの、並列化プログラムに関するデータ再分散に対しても本発明は適用可能である。

【0130】図 26 は、本発明のコンパイラが対象とする並列計算機システムのハードウェア構成の一例を示したものである。同図において、2601 はローカルメモリ、2602 は論理プロセッサエレメント、2603 はネットワーク、2604 は入出力用論理プロセッサエレメント、2605 は入出力用コンソールまたはワークステーションを表す。

【0131】本発明の並列化コンパイラ 100 は、入出力用コンソールまたはワークステーション 2605 において実行され、並列ソースプログラムまたは並列オブジェクトプログラムに変換される。前者の並列ソースプログラムは、さらに、論理プロセッサエレメント 2602 向けのコンパイラにより並列オブジェクトプログラムに変換される。上記並列オブジェクトプログラムはリンクによりロードモジュールに変換され、入出力用論理プロセッサエレメント 2604 を通じて各論理プロセッサエレメント 2602 のローカルメモリ 2601 にロードされ、各論理プロセッサエレメント 2602 により実行さ

れる。実行時における各ロードモジュール間の通信はネットワーク 2603 を通じて行われる。本発明に係る並列化コンパイラは、上記並列計算機システムを有効利用してプログラムを高速化するものである。

【0132】（第2の実施例）次に、本発明の第2の実施例を図27から図33を用いて説明する。第2の実施例では共有メモリを持つ並列計算機を対象としている。図27は、本発明による並列化コンパイラの構成を示したものである。2700は並列化コンパイラである。並列化コンパイラ2700中、ループ並列化解析部2701、データ分散形状決定部2702、プログラム並列化部2703以外は、図1のものと同じであるので説明を省略し、ループ並列化解析部2701、データ分散形状決定部2702、プログラム並列化部2703についてのみ説明する。

【0133】ループ並列化解析部2701は辞書120及び中間語130を入力し、ループの並列性を解析し、ループテーブル150を出力する。この処理の詳細は、ハンス・ジマ、バーバラ・チャップマン著「スーパーコンパイラーズ フォー パラレル アンド ベクトル コンピュータース」アディソン・ウエスリー、1991 (Hans Zima and Barbara Chapman, "Supercompilers for Parallel and Vector Computers", Addison-Wesley, 1991) に説明されている。

【0134】データ分散形状決定部2702は辞書120、中間語130、及びループテーブル150を入力し、データ分散情報140を出力する。ここでは、まず、並列化された各ループに対して、そのループのループ制御変数が、配列のどの次元に出現するかを調べ、次に、ループ並列化によるループ繰り返しのプロセッサへの割り当て方法に対応して、配列要素がどのようなデータ分散方法によってプロセッサに割り付けられるかを調べる。これは、ループ中にある、代入文の左辺の配列添字に、並列化後の各プロセッサへ割り当てられるループ繰り返し範囲を代入することで得ることができる。

【0135】例えば、ループ1のループ繰り返しの1から10、11から20、21から30、31から40回目が各々、プロセッサ0番、1番、2番、3番に割り当てられ、そのループ中で要素数40の配列aが、

$$(i) =$$
 の形で代入文の左辺に出現しているとすると、a(1)からa(10)、a(11)からa(20)、a(21)からa(30)、a(31)からa(40)までが各々、プロセッサ0番、1番、2番、3番によって、値が更新される。即ち、そのループの並列化方法は配列がブロック分散でプロセッサに割り付けられた時と同じになる。後は、上記の方法で得られた分散情報テーブル140を用いて、上述した第1の実施例と同じ方法を適用することで並列化プログラム2720が得られる。

【0136】次に、具体例を用いて説明する。図28

は、ソースプログラム2710を示す図である。これは、図8のソースプログラムから、プロセッサ指示文801、データ分散指示文802、データ再分散指示文804及び810を除いたものである。プロセッサ数はソースプログラム2710では指定されていないが、コンパイルオプションで4と指定されている、とする。

【0137】ループ並列化解析部2701は、ソースプログラム2710を解析し、第1のループネストである文2802から文2806に対してはループiを並列化し、第2のループネストである文2807から文2811に対しては、ループjを並列化する、と決定する。

【0138】データ分散形状決定部2702では、ループ並列化解析部2701が出力したループテーブル150、辞書120、及び中間語130を入力して以下の処理を行なう。まず、プロセッサ数は上記コンパイルオプションで4と指定されているので、第1のループネスト中のループiでは、ループ繰り返しの1から3、4から6、7から9、10から12回目が各々、プロセッサ0番、1番、2番、3番に割り当てられており、この割り当て方法はデータ分散におけるブロック分散と同じである。

【0139】また、ループi中で1次元目の要素数12の配列aが、 $a(i, j) =$ の形で代入文の左辺に出現しているので、a(1, j)からa(3, j)、a(4, j)からa(6, j)、a(7, j)からa(9, j)、a(10, j)からa(12, j)までが各々、プロセッサ0番、1番、2番、3番によって、値が更新される。即ち、配列aの1次元目はブロック分散される。同様にして、第2のループネスト中のループjでは、ループ繰り返しがブロック分散され、ループj中で2次元目の要素数12の配列aが、 $a(i, j) =$ の形で代入文の左辺に出現しているので、配列aの2次元目はブロック分散される。よって、データ分散情報140には、第1のループネストでは、配列aの1次元目がブロック分散され、第2のループネストでは配列aの2次元目がブロック分散される、という情報が設定される。この結果、図9のデータ分散情報が得られる。形状収束部103からプログラム変換部107までの処理は前述した第1の実施例と同じである。

【0140】プログラム並列化部2703での処理も、ほぼ第1の実施例と同じであるが、通信生成ステップ704（図7参照）における `remap send, receive` 通信のかわりにバリア同期を引き起こすルーチンである `barrier` を呼び出し、ループを並列化するために、並列化部分の最初でスレッド生成ルーチン `fork` を、並列化部分の直後でスレッド消滅ルーチン `join` を呼ぶ。

【0141】これらの `fork, join` は、しばしば、並列化ループよりも外側に、しかも、複数の並列化ループに対して1組だけ呼出される。上記 `fork` と `j`

oinによって囲まれる、実際の並列化ループよりも大きい範囲はSPMDリージョンと呼ばれる。このSPMDリージョンの作成処理については、チャウ・ヴェン・ツェン著「コンパイラ最適化技術フォーエブリミネイティングバリエーションシンクロナイゼーション」ピーピーオーピーピー95、第144頁から第155頁、1995(Chau-Wen Tseng, "Compiler Optimizations for Eliminating Barrier Synchronization", PPOPP'95, pp.144-155, 1995)で説明されている。コード生成部109の処理も第1の実施例と同じである。

【0142】図29は、上記の処理の結果得られた並列化プログラム2720を示す図である。文2903と文2904では、図21の、第1の実施例における並列化プログラムにおける文2102及び文2140での再分散ルーチンのかわりに、バリア同期が呼出されている。文2901ではforkが、文2983ではjoinが呼出されている。これらforkとjoinで囲まれた部分がSPMDリージョンである。これら以外は図21のプログラムと同じである。

【0143】図30は従来技術による並列化プログラム20を示す図である。文3003と文3020では、やはり、図23の、第1の実施例における並列化プログラムにおける文2302及び文2320での再分散ルーチンのかわりに、バリア同期が呼出されている。また、文3001ではforkが、文3042ではjoinが呼出されている。これら以外に、図23と異なる点は以下の2つである。

【0144】図23では、配列データが分散メモリに分散されていたため、一つのプロセッサにはそのうちの一部のみが割り付けられていた。そのため、図23における配列aの宣言はa(3, 12)となっていた。しかし、図30は共有メモリ型並列計算機に対するプログラムなので、配列aは分散されず、宣言はa(12, 12)である。

【0145】また、上記と同じ理由により、図23では、各プロセッサは、分散された配列の要素を参照するように、文2311及び文2330で示されるように、ループの繰り返し範囲も縮小されていた。ところが、図30の文3011及び文3030では、各プロセッサは分散されない配列の一部のみを参照するように、プロセッサ番号が格納された変数mypeを用いて、ちょうど自分プロセッサが担当する配列要素のみを参照するようなループ範囲を生成している。

【0146】図31は、従来の並列化によるプロセッサへの計算分担部分を、第1のループネストにおける論理プロセッサ1（以下、P1と略す）の計算分担部分と、第2のループネストにおける論理プロセッサ2（以下、P2と略す）の計算分担部分に注目して表示したものである。3100は12行12列の配列a全体を示す。3100中の数字は、配列要素がアドレス順に並ぶ様子を

示している。左上角から始まる12行3列の部分に、最初の36要素（1～36）のアドレス順が示されている。これにより、1列中の部分は連続アドレスであることがわかる。数字が表示されていない他の部分についても同様である。この本発明による配列要素のアドレスの変更は、図25の場合と同じである。

【0147】4つの角のアドレスが、4, 6, 136, 138番目になる矩形部分3101は、第1のループネストにおける、配列aを(block, *)というデータ分散方法で分散することに対応した、P1の計算分担部分である。また、4つの角のアドレスが、3, 4, 135, 136番目になる網掛け部分3102はプロセッサがキャッシュを持ち、そのキャッシュライン長が配列2要素分であり、配列aの最初の要素がキャッシュライン先頭にある場合に、P1とP0が共有するキャッシュラインを示したものである。この条件の下では、キャッシュラインは、アドレスが奇数番目の配列要素から始まり、それより一つアドレスが大きい配列要素で終わる。したがって、2つのプロセッサは、12個のキャッシュラインを共有し、フォールスシェアリングもその共有キャッシュラインの数だけ発生しやすい。

【0148】4つの角のアドレスが、73, 84, 97, 108番目になる矩形部分3111は、第2のループネストにおける、配列aを(*, block)というデータ分散方法で分散することに対応した、P2の計算分担部分である。この矩形部分3111は、奇数番目のアドレスで始まり、偶数番目のアドレスで終わる単一の連続領域なので、他プロセッサとキャッシュを共有することはない。よって、フォールスシェアリングは第1のループネストにおいてのみ発生する。

【0149】図32は、本発明の並列化によるプロセッサへの計算分担部分を、第1のループネストにおけるP1の計算分担部分と、第2のループネストにおけるP2の計算分担部分に注目して表示したものである。3200は12行12列の配列a全体を示す。3200中の数字は、配列要素がアドレス順に並ぶ様子を示している。左上角から始まる12行3列の部分に、最初の36要素（1～36）のアドレス順が示されている。これにより、太線で囲まれた3行3列中の部分は連続アドレスであることがわかる。数字が表示されていない他の部分についても同様である。

【0150】4つの角のアドレスが、10, 12, 124, 126番目になる矩形部分3201は、第1のループネストにおける、配列aを(block, *)というデータ分散方法で分散することに対応した、P1の計算分担部分である。アドレスが9と10, 45と46, 81と82, 117と118の網掛け部分は、P1とP0が共有する4つのキャッシュラインを示したものである。図31に比べて共有キャッシュラインの数が3分の1になっているので、フォールスシェアリングはそれだ

け、発生しにくい。

【0151】4つの角のアドレスが、73, 79, 102, 108番目になる矩形部分3211は、第2のループネストにおける、配列aを(*, block)というデータ分散方法で分散することに対応した、P2の計算分担部分である。この矩形部分3211は、奇数番目のアドレスで始まり、偶数番目のアドレスで終わる、単一の連続領域なので、他プロセスとキャッシュを共有することはない。よって、フォールスシェアリングは第1のループネストにおいてのみ、発生する。

【0152】今の例では、共有キャッシュラインの数は3分の1になっていたが、一般に、従来技術では、図31において配列aの第2次元の寸法だけキャッシュラインを共有する可能性があり、本発明では、図32において太線で囲まれた部分について1つだけ、即ち、プロセス数だけキャッシュラインを共有する可能性がある。よって、第2次元の寸法がプロセス数を上回れば、後者の方がフォールスシェアリングは発生しにくい。

【0153】図33は、本発明のコンパイラが対象とする並列計算機システムの構成の一例を示したものである。同図において、3301は共有メモリ、3302は論理プロセスエレメント、3303は制御用ネットワーク、3304は入出力用論理プロセスエレメント、3305は入出力用コンソールまたはワークステーションを表す。

【0154】本発明のコンパイラは、入出力用コンソールまたはワークステーション3305において実行され、並列ソースプログラムまたは並列オブジェクトプログラムに変換される。前者の並列ソースプログラムは、さらに、論理プロセスエレメント3302向けのコンパイラにより並列オブジェクトプログラムに変換される。上記並列オブジェクトプログラムはリンクによりロードモジュールに変換され、入出力用論理プロセスエレメント3304を通じて共有メモリ3301にロードされ、各論理プロセスエレメント3302により実行される。論理プロセスエレメント3302の起動、終了などの制御は制御用ネットワーク3303を通じて行われる。

【0155】

【発明の効果】本発明によれば、プロセス1からプロセスjへ送信される部分配列は、プロセス1側でもプロセスj側でも連続アドレスになるので、非連続データを1回の通信で送信するために使われるバッファが不要になり、バッファへのデータのコピー時間が削減されるので、その分、プログラムの実行時間が短縮される。

【0156】また、本発明によれば、各プロセスはデータ分散前の配列全体を宣言して使用する。データ分散の際に、データ領域の確保・開放が不要であり、それにかかる実行時間が短縮される。また、本発明によ

れば、データ分散前の配列全体を宣言して使用しているため、再分散されるデータの内、自プロセスから自プロセスへ送信されるデータは、再分散前後でアドレスが全く変わらないため処理が不要であり、従来、自プロセス内の異なるアドレスへコピーしていたにかかる時間が短縮される。

【0157】また、本発明によれば、プロセス1に割り付けられたデータは、送信先プロセスによって、高々、プロセス数分の非連続な領域に別れるだけなので、プロセス1に割り付けられたデータとプロセスjに割り付けられたデータのアドレスが隣合う回数は、全プロセス数以下となる。したがって、異なるプロセスが同じキャッシュラインを共有する機会が少なくなため、異なるプロセスが同じキャッシュラインをアクセスすることにより発生するフォールスシェアリングの発生する機会が少なく、プログラムの実行時間が短縮される。

【図面の簡単な説明】

【図1】本発明の第1実施例による並列化コンパイラの構成図である。

【図2】配列細分化部の処理を説明するためのフローチャートである。

【図3】配列高次元化部の処理を説明するためのフローチャートである。

【図4】ループ変換部の処理を説明するためのフローチャートである。

【図5】ループピーリング解析処理を説明するためのフローチャートである。

【図6】添字変更部の処理を説明するためのフローチャートである。

【図7】プログラム並列化部の処理を説明するためのフローチャートである。

【図8】ソースプログラムの例を示す図である。

【図9】ソースプログラムに対するデータ分散情報を説明するための図である。

【図10】ソースプログラムに対する辞書を説明するための図である。

【図11】配列細分化による細分化情報を説明するための図である。

【図12】配列高次元化による高次元化辞書を説明するための図である。

【図13】ソースプログラムに対するループテーブルとループ分散範囲テーブルを説明するための図である。

【図14】ソースプログラムに対する配列参照テーブルとピーリングテーブルを説明するための図である。

【図15】ループ範囲細分化後のループ分散範囲テーブルを説明するための図である。

【図16】ループ1のループ範囲細分化後のループテーブルを説明するための図である。

【図17】ループjのループ範囲細分化後のループテー

ブルを説明するための図である。

【図18】ループビバーリング後の中間語をプログラムイメージで表現した図である。

【図19】添字変更後の中間語をプログラムイメージで表現した図である。

【図20】データ分散情報変更後のデータ分散情報を説明するための図である。

【図21】並列化後のプログラムを示した図である。

【図22】データ分散前の配列によるデータ再分散の様子を示した図である。

【図23】従来技術による並列化プログラムを示した図である。

【図24】従来の並列化によるデータ再分散を説明するための図である。

【図25】本発明の第1実施例の並列化によるデータ再分散を説明するための図である。

【図26】本発明の第1実施例のコンパイラが対象とする並列計算機システムの構成の一例である。

【図27】本発明の第2実施例による並列化コンパイラの構成図である。

【図28】ソースプログラムの例を示す図である。

【図29】並列化後のプログラムを示す図である。

【図30】従来技術による並列化プログラムを示す図である。

【図31】従来の並列化によるプロセスへの計算分担部分を説明するための図である。

【図32】本発明の第2実施例の並列化によるプロセスへの計算分担部分を説明するための図である。

【図33】本発明の第2実施例のコンパイラが対象とする並列計算機システムの構成の一例を示す図である。

【符号の説明】

- * 100：並列化コンパイラ
- 101：構文解析部
- 102：データ分散指示文解析部
- 103：形状収束部
- 104：形状種別数判定部
- 105：配列細分部
- 106：配列高次元化部
- 107：プログラム変換部
- 1071：ループ変換部
- 1072：添字変更部
- 108：プログラム並列化部
- 109：コード生成部
- 110、2710：ソースプログラム
- 120：辞書
- 130：中間語
- 140：データ分散情報
- 150：ループテーブル
- 160、2720：並列化プログラム
- 2601：ローカルメモリ
- 2602、3302：論理プロセッサエレメント
- 2603：ネットワーク
- 2604、3304：入出力用論理プロセッサエレメント
- 2605、3305：入出力用コンソールまたはワークステーション
- 2701：ループ並列化解析部
- 2702：データ分散形状決定部
- 2703：プログラム並列化部
- 3301：共有メモリ
- 3303：制御用ネットワーク

【図8】

ソースプログラム 110

```

800:      real a(12,12),b,c
801:!!HPF$ PROCESSORS proc(4)
802:!!HPF$ DISTRIBUTE a(block,*) ONTO proc
803:      do k = 1,10
804:!!HPF$ REDISTRIBUTE a(block,*) ONTO proc
805:      do j = 2,12
806:      do i = 1,12
807:      a(i,j)=a(i,j)+a(i,j-1)*b
808:      enddo
809:      enddo
810:!!HPF$ REDISTRIBUTE a(*,block) ONTO proc
811:      do j = 1,12
812:      do i = 2,12
813:      a(i,j)=a(i,j)+a(i-1,j)*c
814:      enddo
815:      enddo
816:      call update(b,c)
817:      enddo

```

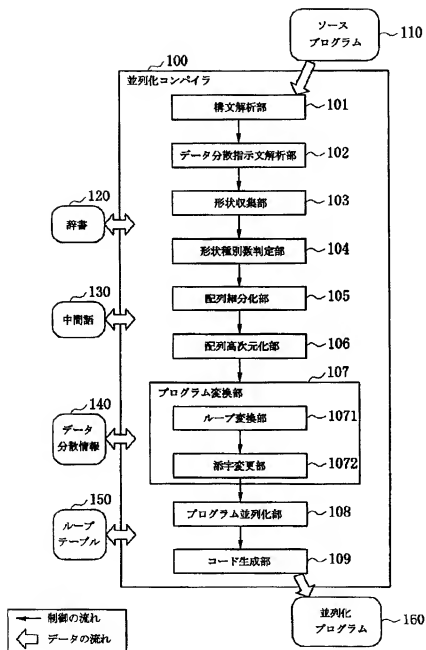
【図11】

並列化情報 1100

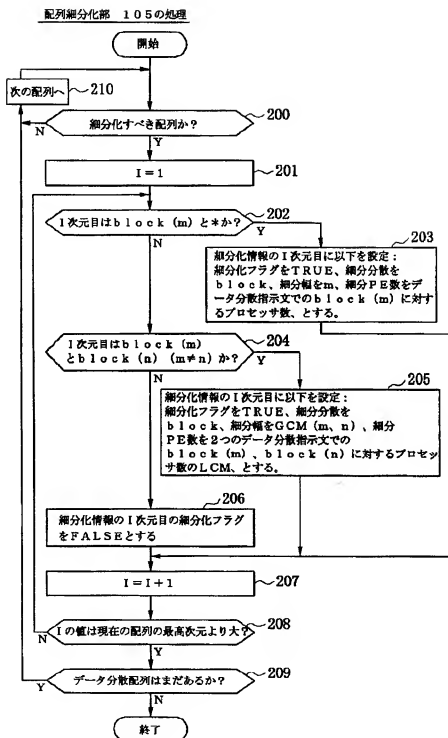
	1100
TRUE	1101
block	1102
3	1103
4	1104
TRUE	1105
block	1106
3	1107
4	1108

【図1】

構成

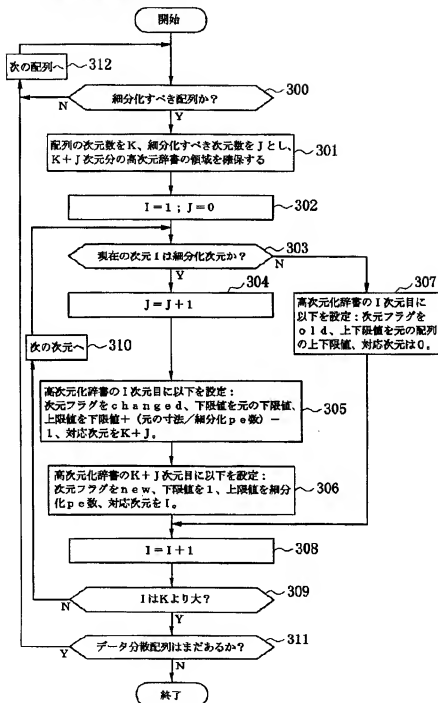


【図2】



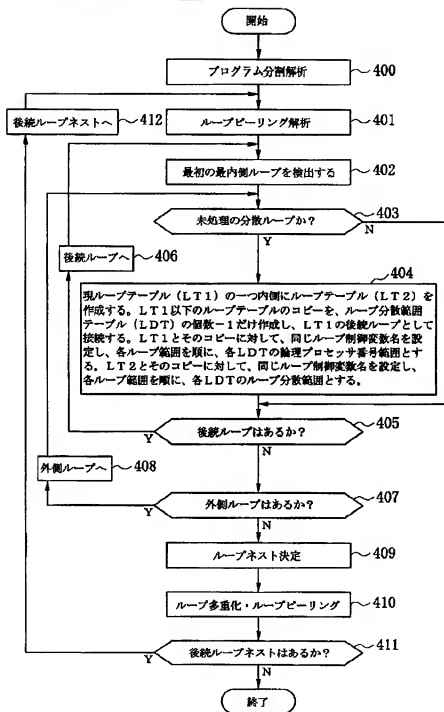
【図3】

配列高次元化部 106の処理



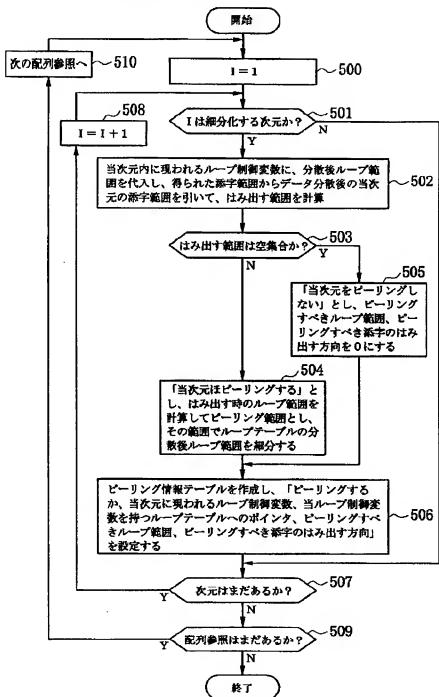
【図4】

ループ変換部 1071 の処理



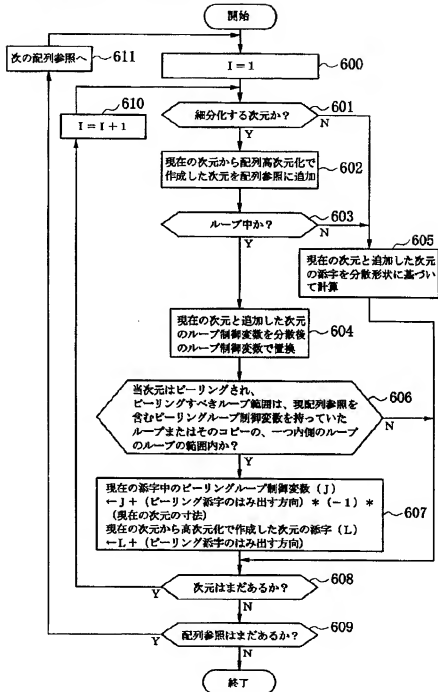
【図5】

ループピーリング解析部 401の処理



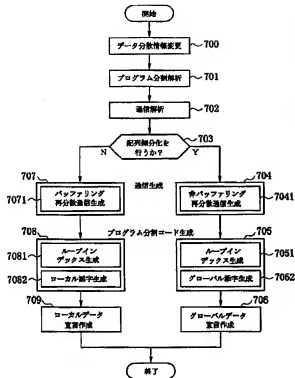
【図6】

添字変更部 1072の処理



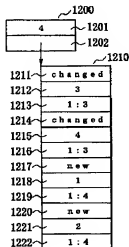
【図 7】

プログラム変換部 108 の処理



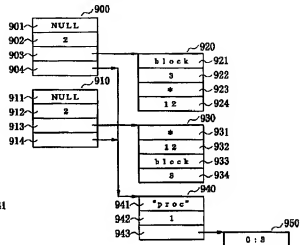
【図 12】

高次元化辞書 1200



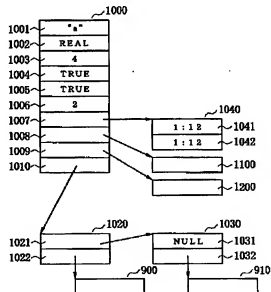
【図 9】

データ分断情報 140



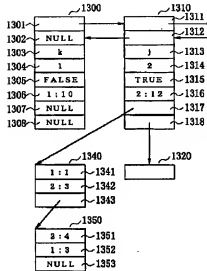
【図 10】

辞書 120



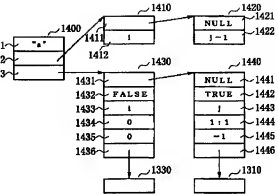
【図13】

ループテーブル



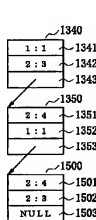
【図14】

配列参照テーブル



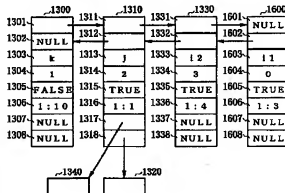
【図15】

ループ範囲細分化後のループ分装範囲テーブル

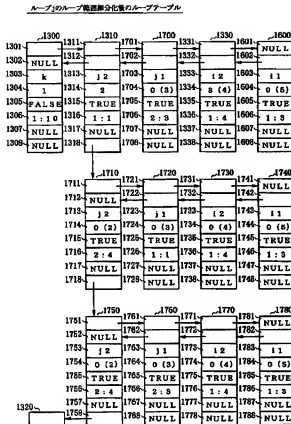


【図16】

ループ1のループ範囲細分化後のループテーブル



【図17】



【図22】

データ分散前の配列によるデータ再分散の様子

2200

0→0	0→1	0→2	0→3
1→0	1→1	1→2	1→3
2→0	2→1	2→2	2→3
3→0	3→1	3→2	3→3

【図18】

ループベリフィケーション後のプログラム

```

1800: do k = 1, 10
1810:   do j2 = 1, 1
1811:     do j1 = 2, 3
1812:       do i2 = 1, 4
1813:         do i1 = 1, 3
1814:           a(i, j) = a(i, j) + a(i, j-1) * b
1815:         enddo; enddo; enddo; enddo
1820:       do j2 = 2, 4
1821:         do j1 = 1, 1
1822:           do i2 = 1, 4
1823:             do i1 = 1, 3
1824:               a(i, j) = a(i, j) + a(i, j-1) * b
1825:             enddo; enddo; enddo; enddo
1830:           do j2 = 2, 4
1831:             do j1 = 2, 3
1832:               do i2 = 1, 4
1833:                 do i1 = 1, 3
1834:                   a(i, j) = a(i, j) + a(i, j-1) * b
1835:                 enddo; enddo; enddo; enddo
1840:               do j2 = 1, 4
1841:                 do j1 = 1, 3
1842:                   do i2 = 1, 1
1843:                     do i1 = 2, 3
1844:                       a(i, j) = a(i, j) + a(i-1, j) * c
1845:                     enddo; enddo; enddo; enddo
1850:                   do j2 = 1, 4
1851:                     do j1 = 2, 4
1852:                       do i2 = 1, 3
1853:                         do i1 = 1, 1
1854:                           a(i, j) = a(i, j) + a(i-1, j) * c
1855:                         enddo; enddo; enddo; enddo
1860:                       do j2 = 2, 4
1861:                         do j1 = 1, 1
1862:                           a(i, j) = a(i, j) + a(i-1, j) * c
1863:                         enddo; enddo; enddo; enddo
1870:                       do j2 = 1, 4
1871:                         do j1 = 2, 3
1872:                           a(i, j) = a(i, j) + a(i-1, j) * c
1873:                         enddo; enddo; enddo; enddo
1880:                       enddo; enddo; enddo; enddo
1881:                     call update(b, c)
1882:                   enddo

```

【図23】

従来技術による並列化プログラム

```

2300: real a(3,12), b, c
2301: do k = 1, 10
2302:   call remap(a, E1, E2)
2310:   do j = 2, 12
2311:     do i = 1, 3
2313:       a(i, j) = a(i, j) + a(i, j-1) * b
2314:     enddo; enddo
2320:   call remap(a, E2, E1)
2330:   do j = 1, 3
2331:     do i = 2, 12
2332:       a(i, j) = a(i, j) + a(i-1, j) * c
2333:     enddo; enddo
2340:   call update(b, c)
2341: enddo

```

【図19】

添字変更後のプログラム

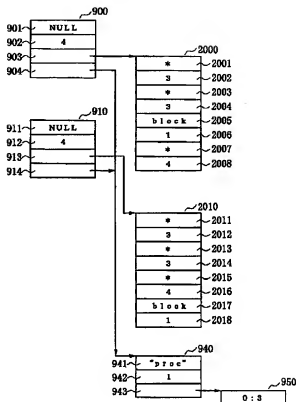
```

1900: real a(3,3,4,4)
1901: do k = 1,10
1910:   do j2 = 1,1
1911:     do j1 = 2,3
1912:       do i2 = 1,4
1913:         do i1 = 1,3
1914:           a(i1,j1,i2,j2)=a(i1,j1,i2,j2)+ a
1915:             a(i1,j1-1,i2,j2)*b
1916:         enddo;enddo;enddo;enddo
1920:       do j2 = 2,4
1921:         do j1 = 1,1
1922:           do i2 = 1,4
1923:             do i1 = 1,3
1924:               a(i1,j1,i2,j2)=a(i1,j1,i2,j2)+ a
1925:                 a(i1,j1+2,i2,j2-1)*b
1926:             enddo;enddo;enddo;enddo
1930:           do j2 = 2,4
1931:             do j1 = 2,3
1932:               do i2 = 1,4
1933:                 do i1 = 1,3
1934:                   a(i1,j1,i2,j2)=a(i1,j1,i2,j2)+ a
1935:                     a(i1,j1-1,i2,j2)*b
1936:                 enddo;enddo;enddo;enddo
1940:             do j2 = 1,4
1941:               do j1 = 1,3
1942:                 do i2 = 1,1
1943:                   do i1 = 2,3
1944:                     a(i1,j1,i2,j2)=a(i1,j1,i2,j2)+ a
1945:                       a(i1-1,j1,i2,j2)*c
1946:                   enddo;enddo
1950:                   do i2 = 2,4
1951:                     do i1 = 1,1
1952:                       a(i1,j1,i2,j2)=a(i1,j1,i2,j2)+ a
1953:                         a(i1+2,j1,i2-1,j2)*c
1954:                     enddo;enddo
1958:                     do i2 = 1,1
1959:                       do i1 = 2,3
1960:                         a(i1,j1,i2,j2)=a(i1,j1,i2,j2)+ a
1961:                           a(i1-2,j1,i2,j2)*c
1962:                       enddo;enddo
1966:                       do i2 = 2,4
1967:                         do i1 = 2,3
1968:                           a(i1,j1,i2,j2)=a(i1,j1,i2,j2)+ a
1969:                             a(i1-1,j1,i2,j2)*c
1970:                         enddo;enddo
1974:                         do i2 = 2,4
1975:                           do i1 = 2,3
1976:                             a(i1,j1,i2,j2)=a(i1,j1,i2,j2)+ a
1977:                               a(i1-1,j1,i2,j2)*c
1978:                           enddo;enddo
1982:                           call update(b,c)
1983:                         enddo
1984:                       enddo
1985:                     enddo
1986:                   enddo
1987:                 enddo
1988:               enddo
1989:             enddo
1990:           enddo
1991:         enddo
1992:       enddo
1993:     enddo
1994:   enddo
1995: enddo

```

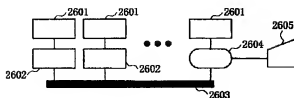
【図20】

変更後のデータ配列図



【図26】

並列計算機システムの一例



【図28】

入カプログラム 2710

```

2800: real a(12,12),b,c
2801: do k = 1,10
2802:   do j = 2,12
2803:     do i = 1,12
2804:       a(i,j)=a(i,j)+a(i,j-1)*b
2805:     enddo
2806:   enddo
2807:   do j = 1,12
2808:     do i = 2,12
2809:       a(i,j)=a(i,j)+a(i-1,j)*c
2810:     enddo
2811:   enddo
2812:   call update(b,c)
2813: enddo

```

【図21】

並列化後のプログラム 160

```

2100: real a(3,3,4,4), b, c
2101: do k = 1, 10
2102:   call non_buf_remap(a, D1, D2)
2110:   do j2 = 1, 1
2111:     do j1 = 2, 3
2112:       do i1 = 1, 3
2113:         a(i1, j1, mype, j2) = a(i1, j1, mype, j2) + &
2114:           a(i1, j1-1, mype, j2) * b
2115:       enddo; enddo; enddo
2120:       do j2 = 2, 4
2121:         do j1 = 1, 1
2122:           do i1 = 1, 3
2123:             a(i1, j1, mype, j2) = a(i1, j1, mype, j2) + &
2124:               a(i1, j1+2, mype, j2-1) * b
2125:           enddo; enddo; enddo
2130:           do j2 = 2, 4
2131:             do j1 = 2, 3
2132:               do i1 = 1, 3
2133:                 a(i1, j1, mype, j2) = a(i1, j1, mype, j2) + &
2134:                   a(i1, j1-1, mype, j2) * b
2135:               enddo; enddo; enddo
2140:               call non_buf_remap(a, D2, D1)
2141:               do j1 = 1, 3
2150:                 do i2 = 1, 1
2151:                   do i1 = 2, 3
2152:                     a(i1, j1, i2, mype) = a(i1, j1, i2, mype) + &
2153:                       a(i1-1, j1, i2, mype) * c
2154:                   enddo; enddo
2160:                   do i2 = 2, 4
2161:                     do i1 = 1, 1
2162:                       a(i1, j1, i2, mype) = a(i1, j1, i2, mype) + &
2163:                         a(i1+2, j1, i2-1, mype) * c
2164:                     enddo; enddo
2170:                     do i2 = 2, 4
2171:                       do i1 = 2, 3
2172:                         a(i1, j1, i2, mype) = a(i1, j1, i2, mype) + &
2173:                           a(i1-1, j1, i2, mype) * c
2174:                       enddo; enddo
2180:                       enddo
2181:                       call update(b, c)
2182:                     enddo

```

【図30】

従来技術による並列化プログラム

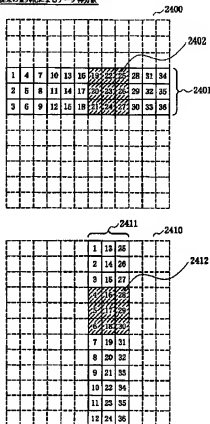
```

3000: real a(12,12), b, c
3001: call fork
3002: do k = 1, 10
3003:   call herrier
3010:   do j = 2, 12
3011:     do i = 3*mype+1, 3*(mype+1)
3012:       a(i, j) = a(i, j) + a(i, j-1) * b
3013:     enddo; enddo
3020:   call herrier
3030:   do j = 3*mype+1, 3*(mype+1)
3031:     do i = 2, 12
3032:       a(i, j) = a(i, j) + a(i-1, j) * c
3033:     enddo; enddo
3040:   call update(b, c)
3041: enddo
3042: call join

```

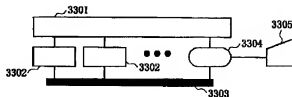
【図24】

従来の並列化によるデータ再分配



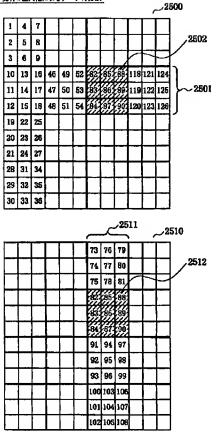
【図33】

並列計算機システムの一例



【図25】

本発明の並列化によるデータ再分配



【図29】

並列化後のプログラム 2720

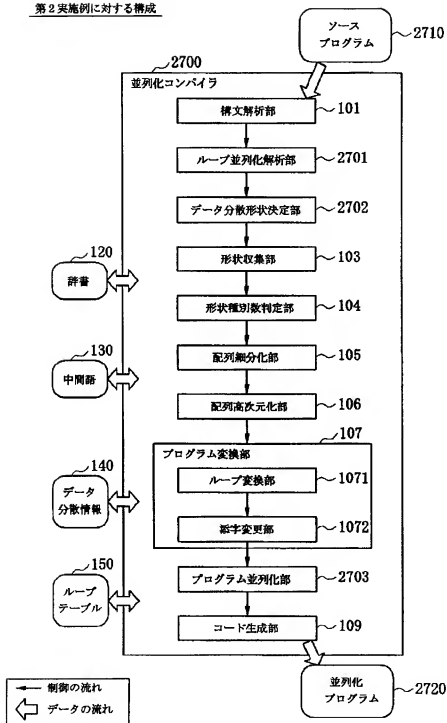
```

2900:real a(3,3,4,4),b,c
2901:call fork
2902:do k = 1,10
2903:  call barrier
2910:  do j2 = 1,1
2911:    do j1 = 2,3
2912:      do i1 = 1,3
2913:        a(i1,j1,mype,j2)=a(i1,j1,mype,j2)+ a
2914:          a(i1,j1-1,mype,j2)*b
2915:      enddo;enddo;enddo
2920:      do j2 = 2,4
2921:        do j1 = 1,1
2922:          do i1 = 1,3
2923:            a(i1,j1,mype,j2)=a(i1,j1,mype,j2)+ a
2924:              a(i1,j1+2,mype,j2-1)*b
2925:          enddo;enddo;enddo
2930:          do j2 = 2,4
2931:            do j1 = 2,3
2932:              do i1 = 1,3
2933:                a(i1,j1,mype,j2)=a(i1,j1,mype,j2)+ a
2934:                  a(i1,j1-1,mype,j2)*b
2935:              enddo;enddo;enddo
2940:              call barrier
2941:              do j1 = 1,3
2950:                do i2 = 1,1
2951:                  do i1 = 2,3
2952:                    a(i1,j1,i2,mype)=a(i1,j1,i2,mype)+ a
2953:                      a(i1-1,j1,i2,mype)*c
2954:                  enddo;enddo
2960:                  do j2 = 2,4
2961:                    do i1 = 1,1
2962:                      a(i1,j1,i2,mype)=a(i1,j1,i2,mype)+ a
2963:                        a(i1+2,j1,i2-1,mype)*c
2964:                    enddo;enddo
2970:                    do i2 = 2,4
2971:                      do i1 = 2,3
2972:                        a(i1,j1,i2,mype)=a(i1,j1,i2,mype)+ a
2973:                          a(i1-1,j1,i2,mype)*c
2974:                      enddo;enddo
2980:                      enddo
2981:                      call update(b,c)
2982:                      enddo
2983:                      call join

```

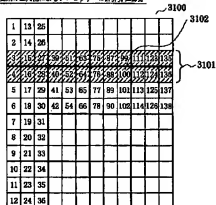
【図27】

第2実施例に対する構成



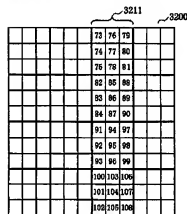
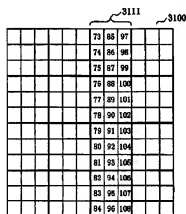
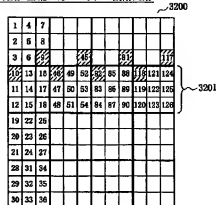
【図31】

従来の並列化によるプロセッサへの割当分岐部分



【図32】

本発明の並列化によるプロセッサへの割当分岐部分



フロントページの続き

(72)発明者 廣岡 孝志
 神奈川県川崎市麻生区王禅寺1099番地 株
 式会社日立製作所システム開発研究所内

Fターム(参考) 5B045 GG11
 5B081 CC28 CC32